Effiziente Algorithmen zur bildbasierten Beleuchtung von Kleidung

Diplomarbeit von Björn Ganster

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik

Bonn, November 2002

Zusammenfassung

Die vorliegende Arbeit stellt eine Reihe von Algorithmen zur Beleuchtung und Berechnung von Schatten vor und vergleicht diese. Für Oberflächen mit Konkavitäten wird in Anlehnung an den Algorithmus von James Stewart [Ste99] ein neues Beleuchtungsmodell entwickelt. Hierbei werden sogenannte Visibility Maps erzeugt, die eine Aufnahme der Szene aus einem Punkt der Geometrie darstellen. Die vorgestellten Algorithmen erlauben insbesondere die Verwendung digital vorliegender Kameraaufnahmen für die Simulation von Beleuchtung (bildbasierte Beleuchtung) unter Berücksichtigung der Selbstabschattung. Dies ermöglicht die Berechnung weicher Schattengrenzen für ausgedehnte Lichtquellen.

Abstract

Several algorithms for computing shadows are discussed and new algorithms for calculating the radiance arriving at selected points of the geometry are proposed. A new model for surface lighting suitable for objects with concavities based on a paper by James Stewart is introduced [Ste99]. In order to accomplish this, so-called visibility maps are used. A visibility map is an image of a scene from the point of view of a vertex of that scene. In particular, the discussed algorithms allow for image-based lighting in combination with self-shadowing. Finally, we demonstrate how to use the visibility maps to calculate shadow boundaries und how to use digital camera pictures for illuminating a scene. This makes it possible to calculate soft shadows for extended light sources.

Einleitung

Die realistische Darstellung von Kleidung ist eine wichtige Komponente bei der Darstellung menschlicher Charaktere in der Computergrafik. Sie erfordert eine Berechnung des Faltenwurfs und die Auswertung eines Beleuchtungsmodells. Während auf dem Gebiet der Berechnung des Faltenwurfs bereits große Fortschritte erzielt wurden, wird für die Beleuchtung meist ein hardware-basiertes Beleuchtungsmodell mit wenigen Punktlichtquellen verwendet, das im Falle von welligen Oberflächen wie Kleidung nur bei sorgfältiger Auswahl der Parameter befriedigende Ergebnisse liefert. Bildbasierte Beleuchtung liefert wesentlich realisterischere Ergebnisse als bei Verwendung von Punktlichtquellen und erlaubt die Simulation der Beleuchtung durch eine beliebige Umgebung aufgrund von fotografierten und computer-generierten Bildern dieser Umgebung. Das Hauptziel dieser Arbeit besteht daher in der Entwicklung eines Algorithmus, der Selbstabschattung bei bildbasierter Beleuchtung berücksichtigt.

Die Bilder, die die bildbasierte Beleuchtung definieren, sollen als *High Dynamic Range (HDR)* Bilder vorliegen. Ein HDR-Bild kann Helligkeiten in einem weitaus größeren Wertebereich speichern als üblicherweise verwendete Bildformate. Diese Bilder werden für eine genaue Auswertung der Beleuchtungsmodelle benötigt, die über sogenannte *Bidirectional reflectance distribution functions (BRDFs)* formuliert wurden. BRDFs haben den Vorteil, dass neben künstlichen, parametrisierten Modellen auch gemessene Reflexionsfunktionen verwendet werden können. Dann entfällt das Einstellen der Parameter völlig, es müssen nur Bilder einer Umgebung und BRDFs der verwendeten Materialien vorhanden sein. Man erhält automatisch realistische Ergebnisse.

Die zentrale Idee des Algorithmus besteht darin, Bilder vom zu beleuchtenden Polygonnetz zu generieren. Diese Bilder werden Visibility Maps genannt und zur Auswertung von Beleuchtungsmodellen verwendet. Die Pixel einer Visibility Map drücken aus, ob für eine Richtung aus Sicht eines Punktes auf dem Polygonnetz eine Lichtquelle von außen den Punkt beleuchten kann oder die Lichtquelle in dieser Richtung durch das Polygonnetz blockiert ist. Durch zusätzliche Informationen, die in die Visibility Map hineingerendert werden, kann eine Beleuchtung durch sogenannte *Environment Maps* durchgeführt werden. Die Environment Maps ermöglichen die oben angesprochene Simulation der Beleuchtung an einem beliebigen Ort. Darüber hinaus können die Environment Maps ausgetauscht und wechselnde Beleuchtung auf derzeitiger Hardware mit einigen Frames pro Sekunde simuliert werden. Außerdem zeigt die Arbeit, wie durch Visibility Maps für Punktlichtquellen Schattengrenzen berechnet werden können.

Obwohl die in der Arbeit besprochenen Algorithmen für Kleidung entwickelt wurden, eignen sie sich für das Rendern beliebiger Polygonnetze. Die Arbeit beschäftigt sich ferner mit den hardware-basierten Ansätzen zur Berechnung von Schatten für Punkt-

lichtquellen und dem Algorithmus von Stewart [Ste99], der einfarbige, umgebende Lichtquellen behandelt.

Da Kleidung meist eine ausgeprägte Struktur hat, bietet es sich an, diese Struktur in Form einer Textur auf die Kleidung aufzutragen. Normale Texturen reichen aber für eine realistische Darstellung meist nicht aus, da das Aussehen des Stoffs vom Einfallswinkel des Lichts und der Position des Betrachters abhängt. Daher wird zur Zeit an der Universität Bonn an der Darstellung von Kleidung durch *Bidirektionale Texturfunktionen (BTFs)* geforscht [Hau02]. BTFs enthalten je nach Beleuchtungs- und Betrachterposition unterschiedliche Texturen und erzielen wesentlich realistischere Ergebnisse als normale Texturen.

Die in dieser Arbeit erwähnten Methoden können anhand eines Programms, das unter [Gan02] heruntergeladen werden kann, nachvollzogen werden. Dort kann man ebenfalls einige Videos herunterladen, die mit dem Programm produziert wurden und die Ergebnisse dieser Diplomarbeit dokumentieren.

In dieser Arbeit wird ein Grundverständnis der Computergrafik vorausgesetzt. Dies betrifft vor allem die Rendering Pipeline von OpenGL und die folgenden Begriffe: Near Clipping Plane, Far Clipping Plane, Depth Buffer, Stencil Buffer. Diese Begriffe werden in der Standardliteratur zum Thema erklärt, beispielsweise [Seg02].

Inhaltsverzeichnis

1 Einführung **1**

1.1 Lichtquellen und Beleuchtungsmodelle 1

1.1.1 Radiometrie 1

1.1.2 Lichtquellen 2

1.1.3 Bidirectional Reflection Distribution Functions (BRDFs) . . . 3

1.1.4 Die Rendering Gleichung 4

1.1.5 Gouraud- und Phong-Modell 5

1.2 Bildbasierte Beleuchtung 9

1.2.1 High Dynamic Range Bilder 9

1.2.2 Environment Mapping 11

1.3 Algorithmen zur Schattenberechnung in Echtzeit 13

1.3.1 Planare Projektion von Schatten 13

1.3.2 zPass Schattenvolumen-Algorithmus 15

1.3.3 zFail Schattenvolumen-Algorithmus 17

1.3.4 Shadow Maps / z-Buffer Schatten-Algorithmus 19

1.3.5 Schlußfolgerungen zu hardware-basierten Schattenalgorithmen 21

1.4 Visibility Cone Algorithmus 22

1.4.1 Aufbau des Visibility Cone Algorithmus 22

1.4.2 Laufzeit von Stewart's Algorithmus zur Ermittlung der Visibility Cones 25

1.4.3 Beleuchtung durch Visibility Cones 26

2 Visibility Map Ansatz **28**

2.1 Algorithmen zur Erzeugung von Visibility Maps 30

2.1.1 Triangle Rasterizer 30

2.1.2 Raytracer 31

2.2 Abbildungen 34

2.2.1 HemiCube Visibility Maps 34

2.2.2 SinglePlane Visibility Maps 35

2.2.3 HemiSphere Visibility Maps 37

2.2.4 Cube Visibility Maps 38
2.3 Probleme an der Near Clipping Plane 39
2.4 Punktlichtquellen und Berechnung von Schattengrenzen mit Visibility Maps 42
2.4.1 Visibility Maps und Punktlichtquellen 42
2.4.2 Berechnung scharfer Schattengrenzen 43
2.4.3 Berechnung von Distance Maps 44
2.4.4 Verwendung von Distance Maps zur Berechnung von Schattengrenzen 47
2.4.5 Näherung für kleine α, β 48
2.4.6 Schattengrenze für Punkte, deren Normale von der Lichtquelle abgewandt ist 48
2.4.7 Beleuchtung an der Schattengrenze 49
2.4.8 Schlußfolgerungen 51
2.5 Beleuchtung von Polygonnetzen mit Visibility Maps und High Dynamic Range Environment Maps 52
2.5.1 HemiCube 53
2.5.2 SinglePlane 54
2.5.3 HemiSphere 54
2.5.4 Bildbasierte Beleuchtung mit Selbstabschattung 55
2.5.5 Beleuchtung mit einer einfarbigen, umgebenden Lichtquelle . 56
2.6 Probleme und Restriktionen der Algorithmen 57

3 Ergebnisse und Ausblick **59**
3.1 Schattengrenzen bei Verwendung von Environment Maps 59
3.2 Berechnung des Lichtaustausches unter Berücksichtigung von BRDFs . 60
3.3 Schlussfolgerungen 61

A Kurzanleitung zum Demonstrationsprogramm **63**

B Performance Messungen **67**

1 Einführung

1.1 Lichtquellen und Beleuchtungsmodelle

1.1.1 Radiometrie

In diesem Abschnitt schaffen wir mit den relevanten Messgrößen für Winkel und Licht die Basis für das Verständnis späterer Beschreibungen des Verhaltens von Licht. Diese Größen werden in der gesamten Arbeit benutzt.

Winkel werden üblicherweise im *Bogenmaß* gemessen. Das Bogenmaß misst einen Winkel als die Länge des auf dem Einheitskreis überdeckten Kreisbogens. Die Einheit des Bogenmaßes ist Radiant (rad), der Vollkreis hat daher einen Raumwinkel von $2\pi\mathrm{rad}$. Der *Raumwinkel* misst die von einer beliebigen Fläche auf der Einheitskugel überdeckte Fläche und wird in Steradiant (sr) gemessen. Da maximal die gesamte Einheitskugel überdeckt sein kann, beträgt der größte Raumwinkel $4\pi\mathrm{sr}$.

Tabelle 1 fasst die wichtigsten Messgrößen für Licht zusammen. Darin werden folgende Größen beschrieben:

- Strahlungsenergie Q
  Licht transportiert *Strahlungsenergie*. Die Energieträger werden Photonen genannt. Jedes Photon trägt die Energie

  $$E = h\nu = h \cdot \frac{c}{\lambda} \tag{1}$$

  wobei

$h \approx 6,62608 \cdot 10^{-34}\mathrm{J} \cdot \mathrm{s}$:	Planckscher Wirkungsquantum (Naturkonstante),
$c \approx 2,99792 \cdot 10^{8}\frac{\mathrm{m}}{\mathrm{s}}$:	die Lichtgeschwindigkeit (Naturkonstante),
ν:	Frequenz des Photons, $[\nu] = \frac{1}{\mathrm{s}}$,
λ:	Wellenlänge des Photons, $[\lambda] = \mathrm{m}$.

- Strahlungsleistung $\Phi = \frac{\partial Q}{\partial t}$
  Strahlungsleistung ist die zeitliche Ableitung der Strahlungsenergie. Sie misst die Helligkeit einer Lichtquelle.

- spezifische Ausstrahlung $E = \frac{\partial \Phi}{\partial A}$
  Die von einer Fläche A ausgesandte Strahlungsleistung Φ heißt *spezifische Ausstrahlung*.

Messgröße	Einheit	Englische Bezeichnung
Strahlungsenergie Q	[J]	Radiant Energy
Strahlungsleistung Φ	[W]	Radiant Flux
spezifische Ausstrahlung E	[W/m^2]	Irradiance
Bestrahlungsstärke B	[W/m^2]	Radiant Exitance
Strahldichte L	[W/(m^2·sr)]	Radiance
Intensität I	[W/sr]	Intensity

Tabelle 1: Messgrößen für Licht

- Bestrahlungsstärke $B = \frac{\partial \Phi}{\partial A}$
  Die von einer Fläche A empfangene Strahlungsleistung Φ heißt *Bestrahlungsstärke*. Sie ist das Gegenstück zur spezifischen Ausstrahlung.

- Strahldichte $L = \frac{\partial^2 \Phi}{\cos\theta \partial\omega \partial A}$
  Für eine Fläche A, die wir aus einem Winkel θ gegen die Normale der Fläche betrachten, ergibt sich für den betrachteten, infinitesimalen Raumwinkel ω die *Strahldichte* aus der obigen Formel. Die Strahldichte ist im Vakuum (und annähernd auch in der Luft) konstant. Daher erscheint ein Objekt unabhängig von seiner Entfernung in der gleichen Farbe.

- Intensität $I = \frac{\partial \Phi}{\partial \varphi}$
  Einer punktförmigen Lichtquelle kann keine spezifische Ausstrahlung zugeordnet werden, da sie keine Fläche hat. Daher misst man bei einer Punktlichtquelle die Strahlungsleistung pro Einheitsraumwinkel Φ, die sogenannte *Intensität*.

1.1.2 Lichtquellen

Da die vorliegende Arbeit sich mit Beleuchtung beschäftigt, ist es wichtig, sich zunächst mit den verschiedenen Typen von *Lichtquellen* vertraut zu machen, die in der Computergrafik verwendet werden. Grundsätzlich gibt es *punktförmige Lichtquellen* und *ausgedehnte Lichtquellen*. Während man bei einer punktförmigen Lichtquelle nur entscheiden muss, ob sie einen Ort beleuchtet und welchen Lichtbeitrag sie liefert, muss bei ausgedehnten Lichtquellen festgestellt werden, welche Teile der Lichtquellen diesen Ort beleuchten und wie groß deren Beitrag zur Beleuchtung ist. Dies erschwert die Auswertung ausgedehnter Lichtquellen so sehr, dass die Beleuchtung mit ausgedehnten Lichtquellen durch Grafikhardware bisher nicht unterstützt wird. Dies ist unbefriedigend, weil in realen Szenen ausgedehnte Lichtquellen überwiegen, die für diffuse Schatten sorgen, während punktförmige Lichtquellen nur scharfe Schatten produzieren.

Wir werden uns trotzdem zunächst mit punktförmigen Lichtquellen beschäftigen, weil wir später eine ausgedehnte Lichtquelle durch mehrere punktförmige Lichtquellen approximieren werden. Die Anzahl der dazu erforderlichen punktförmigen Lichtquellen hängt dabei von der Ausdehnung der Lichtquelle und der erwünschten Genauigkeit der Approximation ab.

Die verschiedenen Modelle für punktförmige Lichtquellen unterscheiden sich überwiegend darin, wie sich ihre Strahlungsleistung auf verschiedene Richtungen verteilt:

- *Punktlichtquelle*: Sie strahlt in alle Richtungen mit gleicher Intensität I und beleuchtet eine Fläche im Abstand r mit der Bestrahlungsstärke $E = \frac{I}{r^2}\cos\theta$, wobei θ der Winkel zwischen der Lichteinfallsrichtung und der Flächennormale ist.
- *Richtungslichtquelle*: Eine Lichtquelle in unendlicher Entfernung ($w = 0$) heißt Richtungslichtquelle. Sie liefert paralleles Licht.
- *Strahler*: Beim Strahler wird Licht überwiegend in eine bestimmte Richtung abgegeben. Die Verringerung bei einer Abweichung um den Winkel ϕ wird durch einen Faktor $\cos^n\phi$ modelliert, wobei n die Breite des Lichtkegels steuert.
- *Goniometrische Lichtquelle*: Bei diesem Lichtquellentyp wird die Intensität für die verschiedenen Lichtquellen durch eine Funktion vorgegeben, die z.B. durch eine Tabelle gegeben sein kann.

1.1.3 Bidirectional Reflection Distribution Functions (BRDFs)

Eine *BRDF* ist eine Materialeigenschaft, die den Anteil des reflektierten Lichts für alle möglichen Kombinationen von Lichteinfallswinkeln und der Richtungen zum Betrachter beschreibt. BRDFs wurden 1977 von Nicodemus definiert [Nic77]. Für einen Punkt $\vec{x}$ ist eine BRDF definiert als:

$$\rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) := \frac{\partial L_o(\vec{x}, \vec{\omega}_o)}{\partial E(\vec{x}, \vec{\omega}_i)} \tag{2}$$

wobei

$\vec{x}$:	betrachteter Punkt, $\vec{x} \in M$,
M:	Zweimannigfaltigkeit,
$\vec{\omega}_i$:	Kugelkoordinaten des einfallenden Lichts,
$\vec{\omega}_o$:	Kugelkoordinaten des reflektierten Lichts,
E:	Bestrahlungsstärke des einfallenden Lichts,
L_o:	Strahldichte des reflektierten Lichts.

Für jeden Punkt $\vec{x}$ ist die BRDF eine Abbildung $\mathbb{R}^4 \rightarrow \mathbb{R}$. Will man BRDFs für die Punkte einer Fläche speichern, erhält man jedoch eine Abbildung $\mathbb{R}^6 \rightarrow \mathbb{R}$. Eine BRDF hat die Einheit $\frac{1}{\mathrm{sr}}$ und sie hat folgende Eigenschaften:

- Reziprozität: die BRDF bleibt bei Vertauschung von Einfalls- und Ausfallswinkel gleich (Grundlage für Raytracing)
- Oft anisotrop: die BRDF ändert sich bei Drehung der Fläche um ihre Normale nicht. In diesem Fall kann die BRDF auf eine Abbildung $\rho : \mathbb{R}^3 \rightarrow \mathbb{R}$ reduziert werden.
- Superposition: überlagertes Licht aus verschiedenen Richtungen beeinflußt sich nicht gegenseitig. Daher kann über die Lichtbeiträge aus verschiedenen Richtungen integriert werden.

Effekte wie *Fluoreszenz* (absorbiertes Licht wird unter anderer Wellenlänge ausgestrahlt), *Phosphoreszenz* (verzögerte Reemission), und Reflexion/Brechung in unteren Schichten des Materials werden von einer BRDF nicht beschrieben. Lafortune beschreibt in [Laf97] ein Verfahren, mit dem aus gemessenen Daten eine BRDF ermittelt werden kann. Wolfgang Heidrich beschreibt in [Hei99] das Gouraud-Modell (Kapitel 1.1.5) als BRDF. Ein weiteres beliebtes Modell ist das Torrance-Sparrow-Modell [Tor66, Tor67]. Heidrich gibt auch für dieses Modell BRDFs an [Hei99].

1.1.4 Die Rendering Gleichung

Die Rendering Gleichung gibt ein Modell für Lichtaustausch an. Sie wurde zuerst von Kajiya formuliert [Kaj86, Hei99]:

$$L_o(\vec{x}, \vec{\omega}_o) = L_e(\vec{x}, \vec{\omega}_o) + \int_{\vec{\omega}_i \in \Omega(\vec{n})} \rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) \cdot L_i(\vec{x}, \vec{\omega}_i) \cdot \cos(\vec{n}, \vec{\omega}_i) d\vec{\omega}_i, \qquad (3)$$

wobei

$\vec{x}$:	betrachteter Punkt,
$\vec{\omega}_i$:	Kugelkoordinaten des einfallenden Lichts,
$\vec{\omega}_o$:	Kugelkoordinaten des reflektierten Lichts,
$L_o(\vec{x}, \vec{\omega}_o)$:	Strahldichte des von $\vec{x}$ ausgesendeten Lichts in Richtung $\vec{\omega}_o$,
$L_e(\vec{x}, \vec{\omega}_o)$:	Strahldichte des von $\vec{x}$ emittierten Lichts in Richtung $\vec{\omega}_o$,
$L_i(\vec{x}, \vec{\omega}_i)$:	Strahldichte des in $\vec{x}$ einfallenden Lichts in Richtung $\vec{\omega}_i$,
$\rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o)$:	BRDF des betrachteten Punktes zwischen den Kugelkoordinaten $\vec{\omega}_i$ und $\vec{\omega}_o$,
$\vec{n}$:	die Normale von $\vec{x}$,
$\Omega(\vec{n})$:	die $\vec{n}$ zugewandte Hemisphäre um $\vec{x}$.

Die von Punkt $\vec{x}$ in Richtung $\vec{\omega}_o$ ausgesendete Strahldichte $L_o(\vec{x}, \vec{\omega}_o)$ setzt sich aus einem reflektierten und einem emittierten Term zusammen. Durch die Verwendung der BRDF können Phosphoreszenz, Fluoreszenz und Reflexion/Brechung in unteren Materialschichten nicht beschrieben werden. Eine vollständige Berechnung des Lichtaustausches innerhalb der Szene wird globale Beleuchtungsberechnung genannt. Während eine exakte Lösung der Gleichung normalerweise unmöglich ist, kann man unter gewissen Annahmen die Gleichung approximativ lösen:

- Radiosity:
  Hier wird der Lichtaustausch zwischen allen Flächen berechnet. Jede Fläche kann gleichmäßig in alle Richtungen Licht emittieren und alle Reflexionen sind diffus. Punktlichtquellen werden nicht unterstützt [Coh85].
- Whitted Raytracing:
  Hierbei wird Licht von einer endlichen Anzahl punktförmiger Lichtquellen ausgesendet und es werden nur spiegelnde Reflexionen berücksichtigt [Whi80].
- Monte Carlo Raytracing:
  hier wird die Rendering Gleichung ausgewertet, indem Strahlen für zufällig ausgewählte Richtungen durch die Szene verfolgt werden [War94].
- hardware-basierte Verfahren mit Schattenberechnung und Environment Mapping:
  Hierbei werden nur Punktlichtquellen unterstützt. Die Szene darf nur wenige spiegelnde Objekte enthalten, da die Berechnung von Interreflexionen sehr aufwändig ist.

1.1.5 Gouraud- und Phong-Modell

In realen Szenen wird meist soviel Licht von anderen Flächen reflektiert, dass Flächen im Schatten einer Lichtquelle in manchen Fällen nur wenig dunkler als direkt beleuchtete Flächen erscheinen. Da die Berechnung dieser Reflexionen durch globale Beleuchtungsrechnung aber äußerst aufwändig ist, verwendet man in der Echtzeit-Computergrafik stattdessen meist einen *ambienten Term*. Dieser wird als konstante Helligkeit auf alle Flächen addiert und soll verhindern, dass Flächen, die nicht direkt von einer Lichtquelle beleuchtet werden, schwarz dargestellt werden:

$$L_{amb} = r_{amb} \cdot c_{amb} \cdot B_{amb}, \tag{4}$$

wobei
L_{amb}: ambienter Strahldichtenterm,
r_{amb}: ambienter Reflexionskoeffizient,
c_{amb}: ambiente Konstante, $c = \frac{1}{\mathrm{sr}}$,
B_{amb}: ambiente Bestrahlungsstärke.

An einem *Lambert-Strahler* wird Licht in alle Richtungen gleichmäßig gestreut. Er stellt damit eine *diffuse Flächenlichtquelle* dar. Die Strahldichte des reflektierten Lichts hängt nur vom Winkel des einfallenden Lichts und dessen Bestrahlungsstärke ab:

$$L_{diff,i} = \begin{cases} r_{diff}(\vec{x}) \cdot g_i(\vec{x}) \cdot I_i \cdot \cos\theta & \text{für } |\theta| \leq \frac{\pi}{2}, \\ 0 & \text{sonst,} \end{cases} \tag{5}$$

wobei

$L_{diff,i}$:	diffuser Strahldichtenterm der i. Lichtquelle,
$\vec{x}$:	betrachteter Punkt,
$r_{diff,i}(\vec{x})$:	diffuser Reflexionskoeffizient,
$g_i(\vec{x})$:	Geometrieterm für die i. Lichtquelle,
I_i:	Intensität der i. Punktlichtquelle,
θ:	Winkel zwischen Flächennormale und Vektor zur Lichtquelle.

Spekulare Reflexion modelliert die Glanzlichter, die für Betrachtungswinkel nahe dem Winkel für spiegelnde Reflexion des Lichts einer Lichtquelle auftreten:

$$L_{spek,i} = \begin{cases} r_{spek}(\vec{x}) \cdot g_i(\vec{x}) \cdot I_i \cdot \cos^m\gamma & \text{für } |\gamma| \leq \frac{\pi}{2}, \\ 0 & \text{sonst,} \end{cases} \tag{6}$$

wobei

$L_{spek,i}$:	spekularer Strahldichtenterm der i. Lichtquelle,
$\vec{x}$:	betrachteter Punkt,
$r_{spek}(\vec{x})$:	spekularer Reflexionskoeffizient,
$g_i(\vec{x})$:	Geometrieterm für die i. Lichtquelle,
I_i:	Intensität der i. Punktlichtquelle,
m:	spekularer Exponent,
γ:	Winkel zwischen reflektiertem Strahl und Richtung zum Beobachter.

Der Geometrieterm $g_i(\vec{x})$ ist wie folgt definiert:

$$g_i(\vec{x}) = \begin{cases} \frac{\mathrm{sr}}{r^2} & \text{falls } \vec{x} \text{ von der Punktlichtquelle } i \text{ beleuchtet wird,} \\ \frac{\mathrm{sr}}{\mathrm{m}^2} & \text{falls } \vec{x} \text{ von der Richtungslichtquelle } i \text{ beleuchtet wird,} \\ 0 & \text{falls sich } \vec{x} \text{ im Schatten der Lichtquelle } i \text{ befindet,} \end{cases} \tag{7}$$

wobei

i:	die betrachtete Lichtquelle ist,
r:	Abstand von $\vec{x}$ zur Lichtquelle,
$\vec{x}$:	der betrachtete Punkt.

Das *Gouraud-Modell* [Gou71] modelliert reflektiertes Licht als Summe eines ambienten, diffusen und spekularen Terms für punktförmige Lichtquellen:

$$L_{out} = L_{amb} + \sum_{i=0}^{n} L_{diff,i} + L_{spek,i}, \tag{8}$$

wobei

L_{out}:	ausgesendete Strahldichte,
L_{amb}:	ambienter Strahldichtenterm,
$L_{diff,i}$:	diffuser Strahldichtenterm für die i. Lichtquelle,
$L_{spek,i}$:	spekularer Strahldichtenterm für die i. Lichtquelle,
n:	Anzahl der Lichtquellen.

Beim Gouraud-Modell wird die Intensität nur an den Vertices berechnet und dazwischen linear interpoliert. Dies ist effizient, aber bei der linearen Interpolation der Normalen bleibt die Länge der Normalen nicht erhalten und daher wird das Beleuchtungsmodell nicht exakt ausgewertet. Insbesondere wird das Glanzlicht einer spekularen Reflexion nicht genau genug berechnet.

Das menschliche Auge reagiert sehr empfindlich auf Änderungen der Helligkeit, da es daraus Informationen über die betrachtete Oberfläche ableitet. Ändert sich bei stückweise linearer Interpolation die Zu- oder Abnahme der Helligkeit, so werden nicht vorhandene Linien sichtbar. Dieser Effekt wird *Mach Band Effekt* genannt [Gou71, Gla95]. Das Phong-Modell [Pho75] verwendet die gleichen Terme wie das Gouraud-Modell, aber es wird für jedes Pixel die interpolierte Normale normiert und das Beleuchtungsmodell pixelweise ausgewertet. Dadurch werden die Mach-Bänder verhindert und die Glanzlichter genauer berechnet. Dies kann in OpenGL durch Pixel Shader geschehen. Normalerweise ist in OpenGL aber nur das Gouraud-Modell verfügbar.

Der diffuse Term lässt sich auch als BRDF angeben. In diesem Fall erhält man eine Konstante:

$$\rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) = k_d,$$

wobei k_d ein diffuser Reflexionskoeffizient ist.

Auch das Phong-Modell für spekulare Reflexion lässt sich als BRDF schreiben:

$$\rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) = k_s \cdot \frac{\cos(\vec{r}, \vec{\omega}_o)}{\cos(\vec{\omega}_i, \vec{n})},$$

wobei

$\vec{x}$:	betrachteter Punkt,
k_s:	spekularer Reflexionskoeffizient,
$\vec{\omega}_i$:	Kugelkoordinaten des einfallenden Lichts,
$\vec{\omega}_o$:	Kugelkoordinaten des reflektierten Lichts,
$\vec{n}$:	die Normale von $\vec{x}$,
$\vec{r}$:	Vektor des reflektierten Lichts.

1.2 Bildbasierte Beleuchtung

In der bildbasierten Beleuchtung werden Lichtquellen durch Bilder einer Szene angegeben. Die Bilder stellen die Umgebung eines Objekts dar und werden daher Environment Maps genannt. Die Pixel einer Environment Map werden nicht als Punktlichtquellen aufgefasst, sondern repräsentieren Raumwinkel und stellen somit ausgedehnte Lichtquellen dar. Bevor näher auf die Beleuchtung durch Environment Maps eingegangen wird, beschäftigen wir uns mit High Dynamic Range Bildern, die uns erlauben, die notwendige Rechengenauigkeit für die erforderlichen Operationen zu erreichen.

1.2.1 High Dynamic Range Bilder

Während in realen Szenen häufig Helligkeitsunterschiede von bis zu 10^{11}:1 auftreten und das menschliche Auge innerhalb einer Szene Helligkeitsunterscheide von 10.000:1 wahrnimmt, können Farbmonitore meist nur Helligkeitsunterschiede von etwa 100:1 darstellen. Daher reicht für die Darstellung auf Monitoren meist das 24-bittige .bmp Bildformat aus, das Rot-, Grün- und Blauanteil einer Farbe in ganzzahligen Schritten zwischen 0 und 255 speichert. Bei manchen Rechenoperationen auf solchen Daten reicht die Genauigkeit dieses Bildformats aber nicht mehr aus, und es stellt sich die Frage, wie Bilder aufgenommen werden können, die einen größeren Helligkeitsunterschied festhalten können.

In der Photographie verwendet man für Bilder mit geringem Unterschied zwischen minimaler und maximaler Helligkeit den Ausdruck *Low Dynamic Range* Bilder. Da wir bei der bildbasierten Beleuchtung im folgenden über Strahldichten integrieren werden, brauchen wir *High Dynamic Range* Bilder. In diesem Bildformat kann eine Helligkeit in einem wesentlich größeren Wertebereich dargestellt werden. Debevec beschreibt in [Deb97], wie aus einer Reihe von Low Dynamic Range Bildern mit normalen Kameras High Dynamic Range Bilder gewonnen werden können.

Wir betrachten kurz, wie Photos aufgenommen werden, um zu verstehen, wie Debevecs Verfahren funktioniert. Bei der Aufnahme eines Bildes reagiert ein Stück Film oder ein Pixel einer elektronischen Kamera auf die Photonen, von denen es getroffen wurde. Je nach aufgenommener Energiemenge fällt diese Reaktion unterschiedlich stark aus. Die Verfärbung bzw. Aufladung nimmt mit der aufgenommenen Lichtenergie zu, der Zusammenhang ist aber nicht linear. Einerseits reagiert der Film auf zu kleine Energiemengen nicht, andererseits tritt ab einer gewissen Energiemenge eine Sättigung auf. Daher verfärbt sich ein Pixel bei doppelter Belichtungszeit nicht doppelt so stark, und bei halber Belichtungszeit nicht unbedingt halb so stark.

Da Bilder mit kurzer Beleuchtungsdauer dunkler sind, kann man auf ihnen vor allem

die hellen Lichtquellen erkennen. Bei Bildern mit längerer Beleuchtungsdauer tritt für diese Lichtquellen dann eine Sättigung auf. Auf diesen Bildern werden die Helligkeitsunterschiede zwischen dunkleren Teilen der Szene deutlich. Wir können durch zusätzliche Bilder also den Messbereich einer Kamera erweitern. Dazu benötigen wir die Funktion, die einer aufgenommenen Lichtenergie eine bestimmte Verfärbung zuordnet. Aus der Umkehrung der Funktion kann für ein Pixel eines Bildes bei bekannter Belichtungszeit die aufgenommene Strahlungsleistung erschlossen werden.

Wir betrachten die Verfärbung

$$Z_{ij} = f(\Phi_i \cdot t_j) \tag{9}$$

eines Pixels i bei Belichtungszeit t_j. f ist eine zu bestimmende Funktion, die die Verfärbung eines Films oder einer elektronischen CCD-Kamera (Charge Coupled Device) aufgrund einer aufgenommenen Energiemenge beschreibt. Ein Pixel wird auf allen Bildern mit Strahlungsleistung Φ_i bestrahlt. Auf einem Bild mit doppelter Belichtungszeit $t_k = 2 \cdot t_j$ ergibt sich für das Pixel also die doppelte aufgenommene Energie. Wir kennen aus dem zweiten Bild die Verfärbung aufgrund der doppelten Energie und haben damit zwei Messpunkte von f. Nachdem wir eine ausreichende Anzahl von Pixeln bei unterschiedlichen Belichtungszeiten ausgewertet haben, können wir jedem Pixel eine einfallende Strahldichte zuordnen, indem wir die Verfärbung in die Umkehrfunktion von f, f^{-1}, einsetzen. Dabei wählen wir für jedes Pixel den Messwert, der der Mitte zwischen minimaler Erregungsenergie und dem Sättigungswert am nächsten liegt. Debevec führt an Gleichung (9) noch einige Umformungen durch, und erhält ein Gleichungssystem, das den Fehler bei der Bestimmung von f minimiert und sicherstellt, dass die Funktion glatt ist.

Zum Speichern eines High Dynamic Range Bildes verwenden wir das .hdr-Format von Greg Ward [War91]. Es speichert Gleitkommazahlen in einem gepackten Format. Eine *Gleitkommazahl* $m \cdot 2^n$ besteht aus dem *Signifikand* m und dem *Exponent* n. Rote, grüne und blaue Farbwerte werden mit einem 8-bittigen Signifikand gespeichert, es wird aber nur der höchste Exponent gespeichert. Dazu müssen die Signifikanden der anderen Zahlen gegebenenfalls an diesen Exponent angepasst werden. Die hellste Farbkomponente wird also mit höchster Genauigkeit gespeichert, während die Signifikanden der anderen Farbkomponenten möglicherweise führende Nullen enthalten. Dadurch geht für diese Zahlen Genauigkeit verloren, denn es stehen nicht alle 8 Bit zur Speicherung des Signifikanden zur Verfügung. Außerdem spielen die kleineren Signifikanden bei der Beleuchtungsrechnung auch eine entsprechend kleinere Rolle. Pro Farbwert werden also ein 8-bittiger Exponent und drei ebenfalls 8-bittige Signifikanden gespeichert und somit 32 Bit für die Codierung einer Farbe benötigt. Im Vergleich zum 24-Bit .bmp-Format werden also nur 8 zusätzliche Bits benötigt, während das Format wesentlich größere Unterschiede von Helligkeiten speichern kann.

1.2.2 Environment Mapping

In der Echtzeitgrafik wird *Environment Mapping* für die Berechnung von Spiegelungen eingesetzt. Die Technik wurde von Jim Blinn entwickelt [Bli76]. Eine *Environment Map* ist ein Panorama, das von einem Punkt eines Objekts, normalerweise von dessen Mittelpunkt, aufgenommen wurde. Beim Rendern der Reflexion wird der Sichtstrahl an der betrachteten Oberfläche gespiegelt und mit der Environment Map geschnitten. Dies liefert für den Strahl eine Texturkoordinate. Diese Berechnung kann entweder pro Pixel oder pro Vertex durchgeführt werden. Bei einer Schnittberechnung pro Vertex wird die Environment Map als Textur verwendet, für die pro Vertex Texturkoordinaten gesetzt werden. Bei einer Schnittberechnung pro Pixel wird für dieses Pixel einfach der berechnete Textureintrag als Reflexion verwendet. Da unterschiedliche Punkte im Raum unterschiedliche Environment Maps haben, kann die Environment Map nur in einem kleinen Umkreis um den Punkt verwendet werden, von dem sie aufgenommen wurde, da die berechneten Spiegelungen sonst nicht passen. Das Objekt, auf das die Spiegelungen aufgetragen werden, muss also klein sein im Vergleich zu seinem Abstand von der Szene.

Environment Maps können auf verschiedene Arten gespeichert werden. Wenn die Umgebung eines Punktes auf einen Würfel projiziert wird, spricht man von einer Cube Map. Wenn die Umgebung eines Punktes auf eine Kugel projiziert wird, erhält man eine Sphere Map. Da eine Cube Map für jede Seite ein Bild enthält, und somit aus sechs Bildern besteht, muss die Szene sechsmal gerendert werden. Dennoch ist dieses Mapping in Hardware inzwischen Standard. Mehrfache Spiegelungen können mit Environment Mapping ausgewertet werden, indem die Interreflexionen durch mehrere Rendering-Zyklen errechnet werden.

Für das Verfahren spielt es keine Rolle, ob die Environment Map ein Bild einer künstlichen Szene oder ein fotografiertes Bild darstellt. Gene Miller, Ken Perlin und paralell Michael Chou und Lance Williams verwendeten eine spiegelnde Kugel, um eine Sphere Map von der Umgebung der Kugel aufnehmen zu können [Deb01]. Indem man diese Sphere Map als Environment Map verwendet, können Computer-generierte Gegenstände in ein fotografiertes Bild eingefügt werden, und die Spiegelung der Umgebung des Objekts kann durch die Environment Map berechnet werden. Die Technik wurde unter anderem in den Filmen „Der Flug des Navigators" und „Terminator II" verwendet.

Man kann vor der Darstellung zusätzlich eine weitere Environment Map berechnen, bei der jedes Pixel für jede Strahlrichtung $\vec{\omega}_o$ das Ergebnis einer Berechnung für diffuse

Reflexion aller einfallenden Strahlrichtungen $\vec{\omega}_i$ enthält. Bei der Darstellung der Szene wird $\vec{\omega}_o$ als Index in diese Datenstruktur verwendet und der diffuse Reflexionsterm zur Spiegelung addiert, wobei die Terme durch den diffusen bzw. spiegelnden Reflexionskoeffizient gewichtet werden. Dieses Modell kann in Echtzeit ausgewertet werden und profitiert von High Dynamic Range Images, da diese eine wesentlich genauere Berechnung des diffusen Terms erlauben. Allerdings kann bei diesem Algorithmus keine Selbstabschattung beachtet werden.

Eine alternative Technik stammt von Debevec [Deb98]. Er teilt eine Szene zunächst in drei Bereiche auf:

- Entfernte Szene: Sie wird allein durch High Dynamic Range Images beschrieben,
- Nahe Szene: hier sind zusätzlich zu einer Beschreibung durch High Dynamic Range Images sowohl Geometrie als auch BRDFs bekannt,
- Künstliche Teile der Szene: sie werden durch Geometrie und BRDFs beschrieben und sollen in ein Bild mit den anderen Bereichen integriert werden.

Wenn im nahen Bereich die BRDF nicht so genau bekannt ist, kann sie durch Vergleich der High Dynamic Range Images und theoretischen Ergebnissen, die aus Geometrie und BRDF ermittelt wurden, genauer bestimmt werden. Debevec verwendet Gregory J. Wards Programm *Radiance* [War94] zur Auswertung der Rendering Gleichung. Dabei handelt es sich um einen stochastischen Raytracer, der für eine möglichst genaue Auswertung der Rendering Gleichung konzipiert wurde und die Berechnung von Interreflexionen und Schatten erlaubt. Die Resultate passen in Bezug auf Beleuchtung und Schattenwurf sehr gut in die Szene, der Raytracer schließt allerdings eine Anwendung in Echtzeit aus. Debevecs Veröffentlichung zeigt außerdem die Vorteile der High Dynamic Range Images gegenüber konventionellen Bildformaten auf.

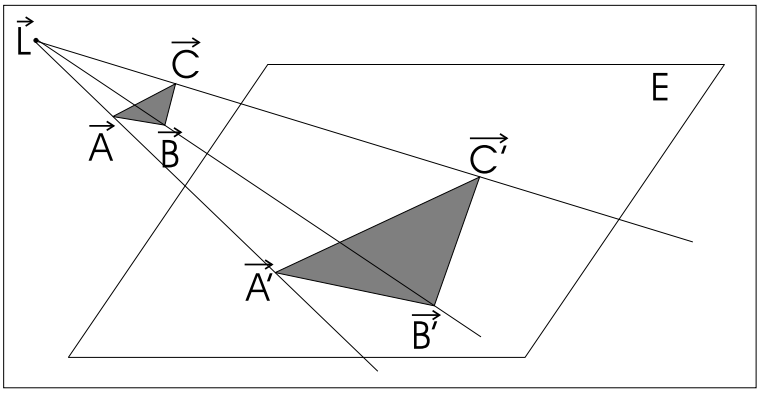

Abb.1: Schatten des $\Delta\vec{A}\vec{B}\vec{C}$ auf E bezüglich der Lichtquelle $\vec{L}$

1.3 Algorithmen zur Schattenberechnung in Echtzeit

Nachdem wir uns in Kapitel 1.1.2 mit Lichtquellen beschäftigt haben, wenden wir uns nun den Standardalgorithmen zu, die den Schattenwurf für Punktlichtquellen berechnen. Dies geschieht relativ detailliert, um die Vor- und Nachteile der Algorithmen aufzeigen zu können. Da eine Strategie zur Simulation ausgedehnter Lichtquellen in der Verwendung vieler Punktlichtquellen besteht, werden wir uns damit beschäftigen, inwiefern sich die Algorithmen zur Beleuchtung durch mehrere Punktlichtquellen eignen.

1.3.1 Planare Projektion von Schatten

Man kann den Schatten eines Objekts auf eine Ebene E berechnen, indem man das Objekt mit der Lichtquelle $\vec{L}$ als Projektionszentrum in die Ebene projiziert (Abb. 1). Jim Blinn verwendet in [Bli88] folgende Matrix zur Projektion des Objekts in die x-y-Ebene für Lichtquellen $\vec{L} = (x_L, y_L, z_L, w_L)$ in homogenen Koordinaten:

$$M_{ShadowProj} = \begin{pmatrix} z_L & 0 & 0 & 0 \\ 0 & z_L & 0 & 0 \\ -x_L & -y_L & 1 & -w_L \\ 0 & 0 & 0 & z_L \end{pmatrix}. \tag{10}$$

Beim Rendern des Schattens ergeben sich dabei zwei Probleme (Abb. 2). Oft soll der Schatten nur auf einen Teil der Ebene fallen, der durch ein Polygon gegeben ist. Dann muss verhindert werden, dass die Schatten über das zu schattierende Polygon hinausragen. Außerdem stellt sich die Frage, in welcher Farbe der Schatten dargestellt werden soll. Eine Lösung für letzeres Problem besteht darin, den Schatten durch halbtransparente Polygone darzustellen. Wenn sich diese jedoch überlappen, entstehen dunkle

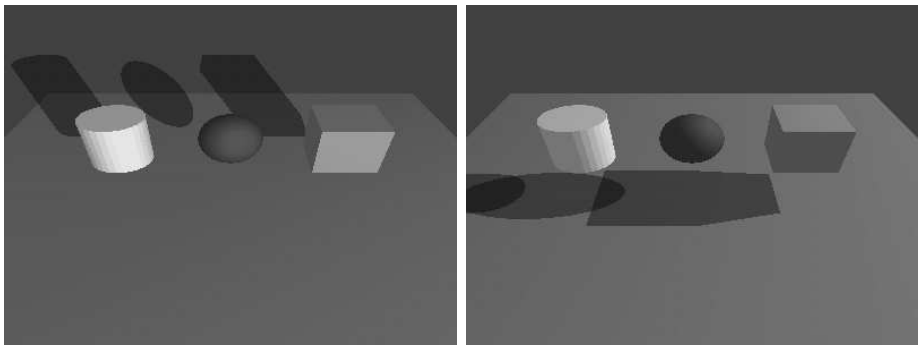

Abb.2: Planar projizierte Schatten ohne Stencil Buffer, links: Schatten ragen über das zu schattierendes Polygon hinaus, rechts: überlappende Schatten

Flecken im Schatten. Mark Kilgard [Kil99] zeigt, wie beide Probleme durch Verwendung des Stencil Buffers [Seg02] verhindert werden können. Dazu speichern wir im Stencil Buffer für jedes Pixel, ob es schattiert werden kann oder nicht. Wir markieren alle Pixel im Stencil Buffer als „nicht schattierbar", die nicht zu einem der zu schattierenden Polygone gehören oder die bereits schattiert wurden. Die restlichen Pixel erhalten die Markierung „schattierbar".

Schattenprojektion mit Hilfe des Stencil Buffers funktioniert wie folgt: Zunächst rendern wir die Polygone der Projektionsebene und setzen ihre Pixel im Stencil Buffer auf „schattierbar". Nun wird die planare Projektion des Schatten werfenden Objekts in die Ebene gerendert. Dabei werden immer nur die Pixel geschrieben, die im Stencil Buffer auf „schattierbar" gesetzt sind. Dadurch kann der Schatten nicht über die Projektionsebene hinausragen. Da der Schatten dieselben Tiefenwerte hat wie die Ebene, deaktivieren wir den Depth Test. Beim Rendern des Schattens werden halbtransparente Polygone verwendet und die Markierung im Stencil Buffer von „schattierbar" auf „nicht schattierbar" gesetzt. Zuletzt wird der Rest der Szene dargestellt, bei deaktiviertem Stencil Test und aktiviertem Depth Test.

Abbildung 3 belegt, wie die Stencil Buffer Erweiterung die überlappenden und doppelten Schatten beseitigt. Wenn man jedem Bit des Stencil Buffers eine Lichtquelle zuordnet, können mehrere Lichtquellen benutzt werden. Die Verdunklung mit Hilfe von Texturen funktioniert zwar auch mit mehreren Lichtquellen, dies ersetzt aber nicht die Auswertung eines richtigen Beleutungsmodells. Insbesondere werden Glanzlichter dadurch nicht ausreichend verdunkelt.

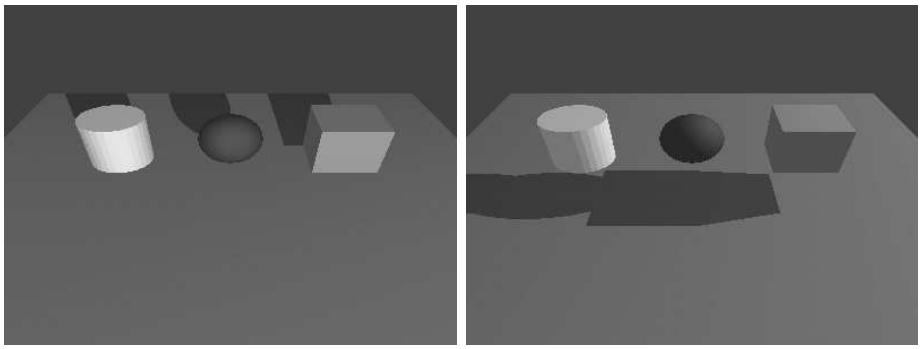

Abb.3: Planar projizierte Schatten mit Hilfe des Stencil Buffers: hier treten die Fehler aus Abbildung 2 nicht mehr auf

1.3.2 *z*Pass Schattenvolumen-Algorithmus

Der im folgenden vorgestellte Algorithmus geht zurück auf Arbeiten von Crow [Cro77] und Erweiterungen von Bergeron [Ber86], Kilgard und Everitt [Kil99, Eve02]. Wie beim vorigen Algorithmus werden Schatten im Stencil Buffer markiert, aber die Berechnung basiert auf *Schattenvolumen*. Ein Schattenvolumen umfasst alle Punkte eines Raumes, die sich im Schatten einer Lichtquelle befinden.

Zur Berechnung der Schattenvolumen werden zunächst die Konturkanten ermittelt. Eine *Konturkante* liegt zwischen einem der Lichtquelle zugewandten und einem der Lichtquelle abgewandten Polygon. Ob eine Fläche der Lichtquelle zu- oder abgewandt ist, stellt man durch Auswertung des Vorzeichens des Skalarprodukts zwischen der Flächennormale und dem Distanzvektor der Fläche zur Lichtquelle fest. Die Konturkanten liegen also zwischen zwei Flächen, bei denen die Vorzeichen dieser Skalarprodukte ungleich sind.

Aus jeder Konturkante wird nun ein *Schattenpolygon* gebildet. Die Schattenpolygone begrenzen wiederum die Schattenvolumina. Die Strahlen, die von den Eckpunkten einer Konturkante entgegengesetzt zur Richtung zur Lichtquelle verlaufen, begrenzen ein Schattenpolygon seitlich. Auf der Seite, die der Lichtquelle zugewandt ist, wird das Schattenvolumen von der Konturkante begrenzt, auf der anderen Seite ragt es ins Unendliche. Für eine Konturkante (E_i, E_{i+1}) ergibt sich also ein Schattenpolygon $\square(E_i, E'_i, E'_{i+1}, E_{i+1})$, wobei E'_i und E'_{i+1} durch Projektion von E_i bzw. E_{i+1} von der Lichtquelle $\vec{L}$ weg ins Unendliche entstehen. Dazu ist eine Translation um $-\vec{L}$ erforderlich, die durch die Translationsmatrix aus [Enc97] beschrieben werden kann. Anschließend wird die Projektion ins Unendliche durch Multiplikation mit folgender Matrix erreicht:

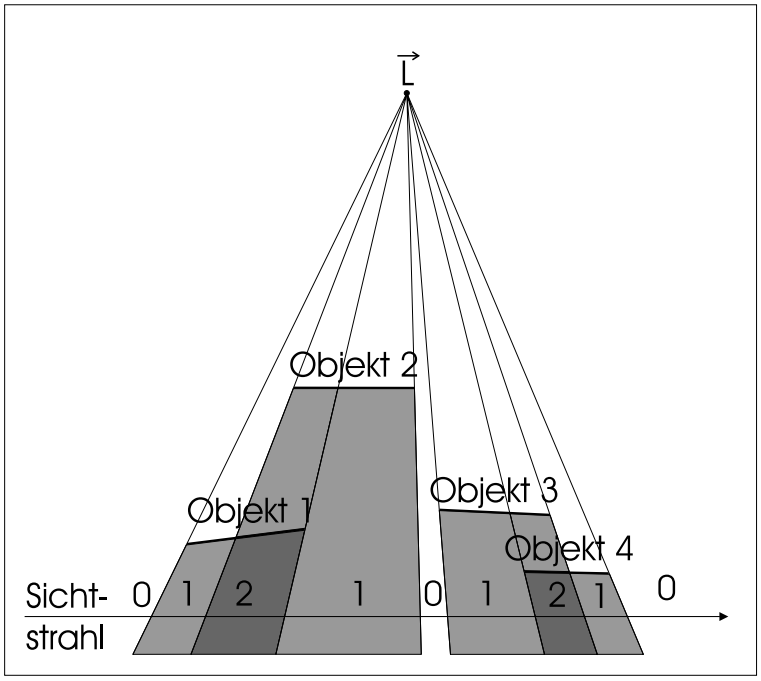

Abb.4: Zählen der Ein- und Austritte eines Sichtstrahles in die Schattenvolumina verschiedener Objekte

$$M_{w=0} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{11}$$

Der Schattenvolumen-Algorithmus versucht festzustellen, ob ein Punkt sich im Schatten einer Lichtquelle befindet oder nicht, indem er die Ein- und Austritte eines Sichtstrahls zum Punkt in die Schattenvolumina mitzählt. Abb. 4 zeigt ein Beispiel. Schneidet der Sichtstrahl die Vorderseite eines Schattenpolygons, dringt er dort in ein Schattenvolumen ein, und der Zähler wird erhöht. Wenn er die Rückseite eines Schattenpolygons schneidet, verlässt er das Schattenvolumen. Ist bei Erreichen des Punktes der Zähler des Sichtstrahls Null, befindet sich der Punkt nicht im Schatten, da jedes Schattenvolumen, in das der Strahl eindrang, auch wieder verlassen wurde. Bei einem Wert ungleich Null befindet sich der Punkt im Schatten, da mindestens ein Schattenvolumen nicht mehr verlassen wurde. Dieser Ansatz funktioniert nur, wenn sich der Startpunkt der Strahlverfolgung nicht in einem Schattenvolumen befindet. Mit einer Erweiterung, die auch in diesem Fall korrekt die Schattenvolumina zählt, befassen wir uns im folgenden Abschnitt.

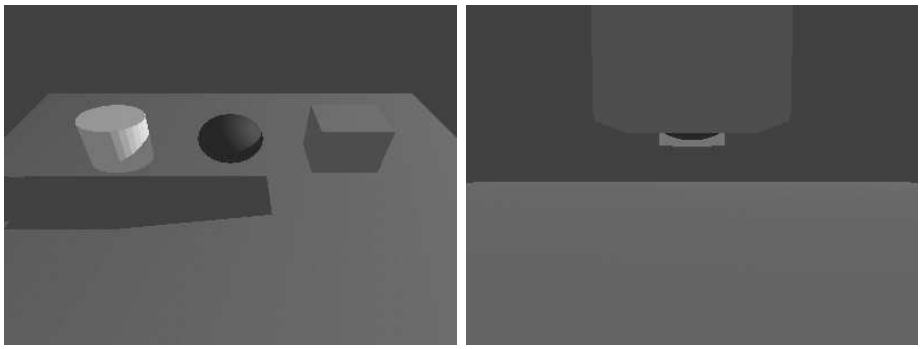

Abb.5: *z*Pass Schattenvolumen-Algorithmus: links: Objekte schattieren sich gegenseitig, rechts: Fehler, wenn Betrachter sich im Schattenvolumen befindet

Der Algorithmus arbeitet wie folgt: Zunächst wird die Szene ohne Lichtquellen gerendert. In OpenGL sorgt in diesem Fall der ambiente Term für eine Restbeleuchtung (Kapitel 1.1.5). Ein wichtiges Ergebnis dieses Durchlaufs ist, dass anschließend der Depth Buffer mit den Tiefenwerten der Pixel initialisiert ist. Nun rendern wir die Schattenpolygone in den Stencil Buffer. Der Stencil Buffer wird dabei von einem Schattenpolygon nur verändert, wenn das Schattenpolygon vor dem betrachteten Punkt liegt. Dies kann ganz normal durch einen Depth Test geprüft werden (daher der Name *zPass Schattenvolumen-Algorithmus*, *z* steht für *z*-Buffer / Depth Buffer). Ist in dem Pixel die Vorderseite des Schattenpolygons sichtbar, wird der Zähler im Stencil Buffer erhöht. Ist hingegen die Rückseite sichtbar, wird der Zähler verringert. Dadurch zählen wir die Schattenvolumen genau wie zuvor beschrieben. Der Color Buffer wird nicht verändert. Im letzten Durchgang rendern wir die Szene erneut bei aktivierter Lichtquelle und ändern den Color Buffer nur in Pixeln, deren Wert im Stencil Buffer Null ist, da sich diese Pixel nicht im Schatten befinden. In den verbleibenden Pixeln bleibt das Ergebnis bei abgeschalteten Lichtquellen aus dem ersten Schritt als Schatten stehen.

Im Gegensatz zur planaren Schattenprojektion erlaubt dieser Algorithmus, dass beliebige Objekte der Szene aufeinander Schatten werfen (Abb. 5, links). Der Algorithmus kann mit mehreren Lichtquellen verwendet werden, indem Schritte 2 und 3 für jede Lichtquelle wiederholt werden und die Beleuchtung additiv durchgeführt wird. Der Algorithmus versagt jedoch, wenn die vorderen Schattenpolygone gegen die Near Clipping Plane stoßen, denn dann wird das Eindringen des Sichtstrahls in ein Schattenvolumen nicht registriert (Abb. 5, rechts).

1.3.3 *z*Fail Schattenvolumen-Algorithmus

Der *z*Pass Schattenvolumen-Algorithmus funktioniert nicht, wenn der Sichtstrahl in einem Schattenvolumen beginnt, denn in diesem Fall müsste der Zähler des Sichtstrahls

mit einem positiven Wert beginnen. Das Zählen der Schattenvolumen scheitert ebenfalls, wenn ein Schattenpolygon gegen die Near Clipping Plane stößt. Diese Schwierigkeiten werden beseitigt, indem man nicht die Schattenvolumina zwischen der Betrachterposition und dem betrachteten Punkt zählt, sondern die Schattenvolumina *hinter* dem betrachteten Punkt.

Wenn sich ein Punkt in einem Schattenvolumen befindet, bedeutet das, dass sich die Rückseite des Schattenvolumens aus Sicht des Betrachters hinter dem Punkt befindet. Während wir vorher festgestellt haben, dass der Sichtstrahl von der Betrachterposition zum Punkt nur die Vorderseite, aber nicht die Rückseite eines Schattenvolumens schnitt, registrieren wir nun, dass sich zwar die Rückseite des Schattenvolumens hinter dem Punkt befindet, nicht aber dessen Vorderseite. In beiden Fällen finden wir heraus, dass sich der Punkt im Schatten befindet. Der neue Test funktioniert aber auch, wenn der Sichtstrahl in einem Schattenvolumen startet, denn in diesem Fall wird ja durch die Rückseite des Schattenvolumens festgestellt, dass wir uns im Schatten der Lichtquelle befinden.

Im Gegensatz zum vorigen Algorithmus werden Schattenpolygone also dann gezählt, wenn für sie der Depth Test *fehlschlägt*, denn dann befindet sich das Schattenpolygon *hinter* dem betrachteten Pixel. Deshalb hat der Algorithmus den Namen *zFail Schattenvolumen-Algorithmus*. Nach dem ersten Renderdurchgang wird der Zähler für rückseitige Schattenpolygone erhöht, und anschließend für die Vorderseiten der Schattenpolygone verringert. Der Rest des Algorithmus funktioniert wie beim zPass Schattenvolumen-Algorithmus.

Die Kamera muss mindestens den Abstand der Near Clipping Plane von der darzustellenden Geometrie einhalten, da sonst Zählfehler auftreten, weil der Depth Test hinter geclippter Geometrie nicht fehlschlagen kann. Diese Konvention ist generell üblich, da man sonst durch Geometrie hindurch schauen kann, wenn man sich nahe genug auf sie zu bewegt. Abgesehen davon sind die Probleme an der Near Clipping Plane also behoben. Damit der Stencil Buffer Test auch für z-Werte hinter der Far Clipping Plane durchgeführt wird, wird die Far Clipping Plane ins Unendliche verlegt. Betrachten wir dazu die mit dem OpenGL-Befehl `glFrustum` definierte Matrix:

$$M_{frustum} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}, \tag{12}$$

$$\lim_{f \to \infty} M_{frustum} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{pmatrix}, \tag{13}$$

für
- l: linke Fensterbegrenzung,
- r: rechte Fensterbegrenzung,
- t: obere Fensterbegrenzung,
- b: untere Fensterbegrenzung,
- n: Entfernung zur Near Clipping Plane,
- f: Entfernung zur Far Clipping Plane.

Die Tiefenauflösung im Depth Buffer für unendlich entfernte Objekte ist zwar sehr schlecht, aber wir wollen ja nur Clipping vermeiden. An der Far Clipping Plane ist der Verlust an Präzision laut Cass Everitt [Eve02] hingegen akzeptabel.

Beim zPass Schattenvolumen-Algorithmus wurden die Schattenvolumen nicht im Objektinneren geschlossen, da sich diese Polygone im Inneren des Polygonnetzes befinden und für sie daher der Depth Test immer fehlschlägt. Beim zFail Schattenvolumen-Algorithmus können wir diese Polygone hingegen nicht mehr vernachlässigen. Die Schattenvolumen können im Objektinneren aber ohne Mehraufwand geschlossen werden, indem wir im ersten Durchlauf alle der Lichtquelle zugewandten Polygone zusätzlich in den Stencil Buffer rendern. Die Schattenvolumen müssen aber auch an der Far Clipping Plane geschlossen sein. Dazu projizieren wir die der Lichtquelle abgewandten Polygone ins Unendliche, wie wir das bei den Konturkanten getan haben, und rendern sie in den Stencil Buffer. Dies verursacht einen geringen Mehraufwand. Abb. 6 zeigt, dass Schatten vom zFail Schattenvolumen-Algorithmus auch dann richtig berechnet werden, wenn sich der Betrachter im Schatten der Lichtquelle befindet.

Da die meisten erhältlichen Grafikkarten inzwischen einen Stencil Buffer unterstützen, ist die Hardware-Unterstützung für diesen Algorithmus unproblematisch. Tabelle 2 vergleicht die Geschwindigkeit der oben vorgestellten Schattenalgorithmen.

1.3.4 Shadow Maps / z-Buffer Schatten-Algorithmus

Bei diesem Algorithmus wird für jede Lichtquelle ein Bild der Szene aufgenommen und der z-Buffer als Textur gespeichert. Dadurch entsteht die sogenannte *Shadow Map*. Anschließend wird die Szene aus der Betrachterposition gerendert. Um nun zu überprüfen, ob ein Punkt $\vec{P}$ im Schatten der Lichtquelle liegt, wird $\vec{P}$ in die Shadow Map transformiert und gegen den dort eingetragenen Tiefenwert getestet. Hat $\vec{P}$ einen

Algorithmus	Framerate
Ohne Schatten	378 fps
Planare Projektion	192 fps
Planare Projektion mit Stencil Buffer	192 fps
zPass Shadow Volumes	74 fps
zFail Shadow Volumes	56 fps

Tabelle 2: Laufzeitvergleich der Schattenalgorithmen für eine Szene mit 2602 Dreiecken, gemessen auf einem Athlon 1800 MHz mit einer GeForce4 TI 4200 bei einer Auflösung von 1024x768 Pixeln mit 24 Bit Farbtiefe

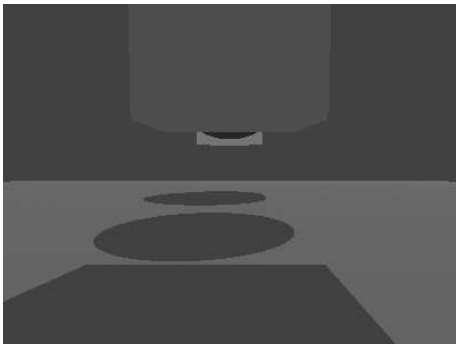

Abb.6: Der zFail Schattenvolumen-Algorithmus funktioniert auch dann, wenn sich der Betrachter im Schatten der Lichtquelle befindet

größeren Tiefenwert als in der Shadow Map vermerkt, ist er aus Sicht der Lichtquelle verdeckt und befindet sich daher im Schatten. Anderenfalls muss die Lichtquelle bei Darstellung des Punkts aktiviert sein. Je nach vorhandener Hardware kann dieser Test entweder für jedes Pixel durchgeführt werden oder man beschränkt sich auf die Eckpunkte eines Polygons. Während der Test nur auf Hardware ab dem Niveau einer GeForce3 pixelweise durchgeführt werden kann, reicht für den Test der Eckpunkte einfachste Computergrafik Hardware aus.

Wenn der Verdeckungstest auf der Grafikkarte durchgeführt wird, wird die CPU nach der Generierung der Szene kaum noch belastet, denn beim Rendern der Depth Map kann eine Display List für den zweiten Durchgang generiert werden. Eine *Display List* speichert OpenGL-Treiberaufrufe als Liste ab, die dann beim Aufruf der Liste effizienter durchgeführt werden können, als wenn jede OpenGL-Funktion einzeln aufgerufen wird. Nach dem Einstellen der erforderlichen Parameter wird dann nur noch die Display List aufgerufen, und die Grafik-Hardware übernimmt den Rest der Arbeit.

Seitdem Lance Williams in [Wil78] Shadow Maps einführte, wurden eine Reihe interessanter Erweiterungen entwickelt. Wenn die Perspektive der Shadow Map schlecht zur Betrachterperspektive passt, werden für die Schatten des Bildes nur wenige Pixel der Depth Map ausgewertet, was zu Aliasing führt. William T. Reeves und Marc Stamminger zeigen hierzu Lösungen auf [Ree87, Sta02]. Eine weitere Schwierigkeit liegt darin, dass eine Shadow Map, die mit einer linearen Abbildung erzeugt wurde, nicht den ganzen Halbraum eines Bildes abdeckt. Um den gesamten Raum um einen Punkt abzubilden, werden dann wie bei einer Cube Environment Map sechs Bilder benötigt. Die Szene müsste nicht nur sechsmal gerendert werden, sondern es müsste auch eine aufwändige Fallunterscheidung durchgeführt werden, um zu entscheiden, in welches Bild ein Pixel transformiert werden muss. Vor allem wird Antialiasing durch die vielen Fallunterscheidungen fast unmöglich. Stefan Brabec beschreibt in [Bra02] ein Verfahren, das mit zwei Bildern durch dual-parabolische Mapping den gesamten Halbraum um einen Punkt abdeckt.

1.3.5 Schlußfolgerungen zu hardware-basierten Schattenalgorithmen

Bei der Schatten-Projektionstechnik bleibt unklar, welche Farbe dem Schatten zugeordnet werden soll. Die halbtransparenten Schattenpolygone lösen diese Aufgabe selbst mit Stencil Buffer nur unbefriedigend, da für den Schatten kein Beleuchtungsverfahren angewendet wird, sondern die Fläche dort einfach verdunkelt wird. Wie bei allen vorgestellten Algorithmen müssen für jede Lichtquelle große Teile der Szene neu gerendert werden.

Auch wenn die Hardware-beschleunigten Algorithmen gute Resultate für Punktlichtquellen liefern, sind sie für ausgedehnte Lichtquellen nur schlecht geeignet. Ausgedehnte Lichtquellen haben einen Halbschatten, in dem die Helligkeit je nach Form der Lichtquelle nichtlinear zunehmen kann. Zur Nachahmung der Halbschatten können mehrere Lichtquellen verwendet werden. Dann werden anstelle einer Schattengrenze mehrere Schattengrenzen dargestellt, die aber für einen Halbschatten noch zu scharf sind. Die hardware-beschleunigten Verfahren liefern daher für die Simulation ausgedehnter Lichtquellen nur unzureichende Ergebnisse.

1.4 Visibility Cone Algorithmus

Die bisher betrachteten Ansätze berechnen Schatten von Punktlichtquellen für jedes Bild komplett neu. Der Algorithmus von Stewart hingegen berechnet vor der Darstellung eines Polygonnetzes eine Lösung der Rendering Gleichung ausgehend von der Geometrie des Polygonnetzes [Ste99]. Dabei wird ein sehr einfaches Lichtquellenmodell angenommen: die Lichtquelle umgibt das Polygonnetz von allen Seiten, ist einfarbig, und das Material des Polygonnetzes wird als rein diffus reflektierend angenommen. Die Lösung kann anschließend angezeigt werden, ohne dass weitere Berechnungen zur Beleuchtung erforderlich wären. Die ermittelte Lösung gilt jedoch nur für statische Geometrie. Wenn sich die Geometrie verändert, muss eine neue Lösung der Rendering Gleichung ermittelt werden, dies kann jedoch nicht in Echtzeit geschehen.

1.4.1 Aufbau des Visibility Cone Algorithmus

James Stewart geht bei der Schattenberechnung von den Knoten eines Polygonnetzes aus und berechnet für jeden Knoten eines Polygonnetzes ein oder mehrere Polygone, die sogenannten *Visibility Cones* (*Sichtbarkeitskegel*). Das Innere eines solchen Polygons repräsentiert die Richtungen, in denen eine Lichtquelle aus Sicht des Punktes nicht durch das Polygonnetz blockiert wird. Die Verdeckungsinformation in einem Visibility Cone wird dann zur Auswertung von Lichtquellen benutzt.

Zur Berechnung eines Visibility Cone wird zunächst das Polygonnetz für jede von mindestens sechs Grundrichtungen in Scheiben konstanter Dicke aufgeteilt, die zur jeweiligen Grundrichtung senkrecht stehen. Jede Scheibe wird daraufhin in eine Ebene projiziert. Punkte, die zu einem gemeinsamen Dreieck gehören, werden durch eine Kante verbunden.

Wenn Punkte fast übereinander liegen, werden sie durch die Projektion so abgebildet, dass sie anschließend sehr nahe beieinander in der Projektionsebene liegen. Dies führt zu Verzerrungen der Konturen und schlimmstenfalls versagen bestimmte Algorithmen. Es wurden für das Verfahren also Algorithmen gewählt, die auch bei kleinen Strukturen korrekte Vorhersagen liefern. Stewart schlägt als Alternative vor, zu nahe beieinander liegende Punkte zu entfernen.

Nun werden für jede Ebene zweidimensionale Visibility Cones bestimmt. Ein zweidimensionaler Visibility Cone ist die Schnittmenge eines dreidimensionalen Visibility Cone mit einer Ebene. Abb. 7 zeigt die Visibility Cones der Punkte P und Q. Wir beginnen, indem wir die konvexe Hülle der Punkte bestimmen [Meh99] und triangulieren die Punktmenge. Die konvexe Hülle enthält einige Kanten, die im Polygonzug

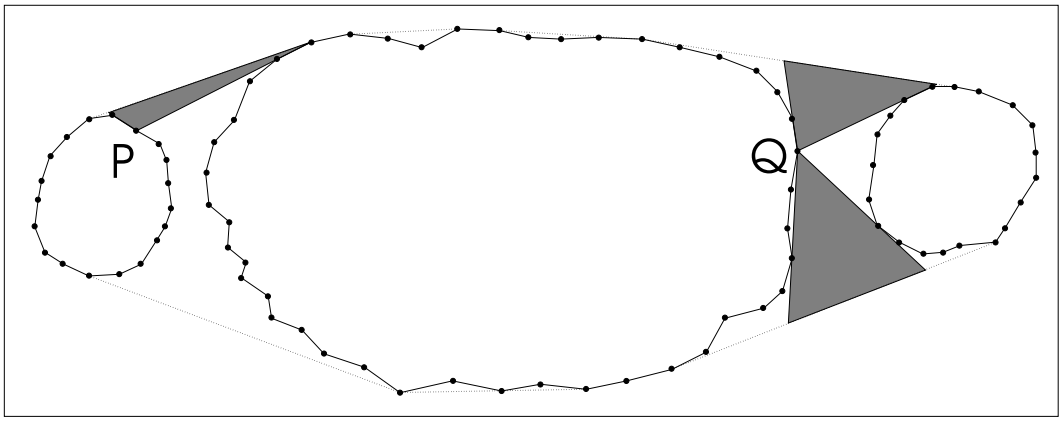

Abb.7: Schnittebene durch die Kleidung einer Person, mit leichten Falten. Die konvexe Hülle ist gestrichelt dargestellt, die Fenster der Punkte P und Q grau.

nicht vorhanden sind. Sie werden Fenster genannt, weil durch diese Kanten Licht ins Innere einer Falte fallen kann. Eine Falte ist eine Menge $\{p_0; ...; p_n\}$ von Punkten des Polygonzuges, die alle dieselben Fensterkanten haben. Alle Punkte, die nicht auf der konvexen Hülle liegen, werden Falten zugeteilt. Im triangulierten Graph gehören die Eckpunkte aller Dreiecke zur selben Falte, die benachbart sind und nicht durch den Polygonzug getrennt sind. Wir starten zur Feststellung dieser Punkte eine Tiefensuche von den Fensterkanten aus. Da die Ebene mehrere Polygonzüge enthalten kann, kann ein Punkt aber auch mehrere Fenster haben (Abbilung 7, Punkt Q). In diesem Fall kann aus jedem Fenster ein zweidimensionaler Visibility Cone für einen Punkt entstehen. Folgende Fälle sind möglich:

- Wenn der betrachtete Punkt auf der konvexen Hülle liegt, aber nicht zu einem Fenster gehört, ergeben sich seine zweidimensionalen Visibility Cones direkt aus den benachbarten Kanten der konvexen Hülle.

- Wenn der betrachtete Punkt zu einer Fensterkante gehört, kann er für jede Falte, zu der er gehört, zusätzliche zweidimensionale Visibility Cones haben (Punkt Q in Abb. 7).

- Verdeckt keine Kante des Polygonzuges das Fenster eines Punktes P in einer Falte, bildet das Fenster den zweidimensionalen Visibility Cone des Punktes (Punkt P in Abb. 8).

- Wenn mindestens eine Kante in die Verbindung zur Fensterkante hineinragt, schränkt sie den Visibility Cone ein (Punkt Q in Abb. 8).

- Wenn das Fenster durch Kanten der Falte vollständig verdeckt ist, ist der zweidimensionale Visibility Cone dieses Fenster leer (Punkt R in Abb. 8).

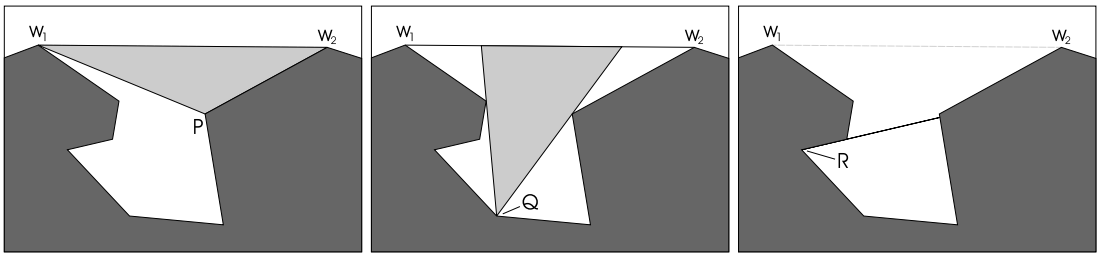

Abb.8: Zweidimensionale Visibility Cones verschiedener Punkte

In all diesen Fällen gilt: Der zweidimensionale Visibility Cone ist für jedes Fenster eines Punkts durch die erste Kante des kürzesten Weges des betrachteten Punktes zu dessen Fensterkanten gegeben. Anstatt für jeden Punkt den kürzesten Weg zu den Endkanten des Fensters zu berechnen, wird für jede der Endkanten ein Baum errechnet, der die kürzesten Pfade zu den Knoten der Falte enthält. Hierbei handelt es sich nicht um kürzeste Wege entlang der Kante, sondern die Wege verlaufen durch den freien Raum, der von den Kanten begrenzt wird.

Nachdem für alle Punkte in allen Ebenen und für alle Richtungen zweidimensionale Visibility Cones berechnet wurden, müssen daraus dreidimensionale Visibility Cones ermittelt werden. Da manche Punkte jedoch mehrere dreidimensionale Visibility Cones haben, muss zuerst festgestellt werden, welche zweidimensionalen Visibility Cones einen gemeinsamen dreidimensionalen Visibility Cone formen. Um für zwei zweidimensionale Visibility Cones zu testen, ob sie zu demselben dreidimensionalen Visibility Cone gehören, bildet man die Schnittgerade der Ebenen, in denen die Kegel liegen. Liegt sie in beiden zweidimensionalen Kegeln, schneiden sich diese und gehören zu demselben dreidimensionalen Kegel. Der dreidimensionale Visibility Cone wird nun aus der umlaufenden Sortierung seiner Punkte um den Schwerpunkt des Polygons ermittelt.

Abb. 9 zeigt die Visibility Cones der Punkte P und Q. Links oben sehen wir zwei zweidimensionale Visibility Cones, die rechts oben zu einem dreidimensionalen Visibility Cone zusammengefasst werden. Links unten sehen wir vier zweidimensionale Visibility Cones. Obwohl sich nicht alle Visibility Cones berühren, formen sie einen dreidimensionalen Visibility Cone. Das Beispiel verdeutlicht den Hauptnachteil der Visibility Cones: es stehen keine Daten über den Visibility Cone außerhalb der zweidimensionalen Visibility Cones zur Verfügung. In diesem Fall führt der Ansatz, die Punkte einfach zu verbinden, zu einem Fehler. Obwohl der untere Visibility Cone unterbrochen ist, verbindet der Algorithmus einfach die Eckpunkte an der mit einem Fragezeichen markierten Stelle, da der Algorithmus keine Daten über den Verlauf des Visibility Cones zwischen diesen Punkten hat. Während ohnehin unklar ist, wie die Visibility Cones außerhalb der berechneten Eckpunkte verlaufen, wird in diesem Fall

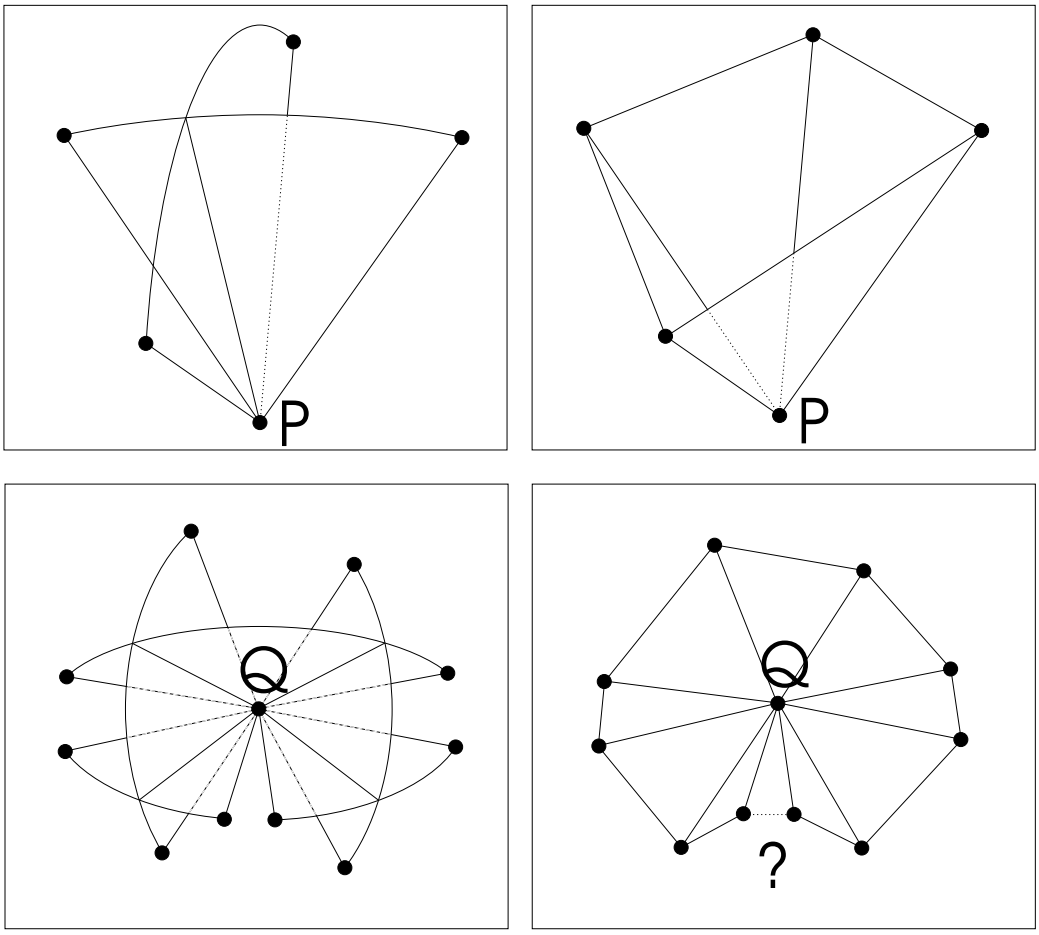

Abb.9: Erzeugung dreidimensionaler Visibility Cones aus zweidimensionalen Visibility Cones

sogar ein Fehler in Kauf genommen. Bei der umlaufenden Sortierung der Punkte des Visibility Cones können weitere Fehler auftreten, weil die umlaufende Sortierung nicht notwendigerweise mit der Reihenfolge der Punkte des Polygons übereinstimmt.

1.4.2 Laufzeit von Stewart's Algorithmus zur Ermittlung der Visibility Cones

Nimmt man an, dass sich die Punkte des Polygonnetzes gleichmäßig über das Volumen verteilen, erhielte man bei n Punkten, die durch $\sqrt{n}$ scheibenförmigen Volumina mit identischen Abmessungen unterteilt sind, für jedes Volumen $\sqrt{n}$ Punkte. Die Punkte jedes Volumens werden in die scheibenförmige Grundfläche des Volumens projiziert. Sei m die Zahl der Punkte, die zu einer Ebene gehören, $m \approx \sqrt{n}$. Durch einmalige Iteration über die Punktmenge kann jeder Punkt mit Zeitaufwand $O(n)$ einer Ebene zugeteilt werden. Die Berechnung der konvexen Hülle der Punkte einer Ebene benötigt $O(m \log m)$ Zeiteinheiten. Dabei können den Punkten auf der konvexen Hülle bereits

zweidimensionale Visibility Cones zugeordnet werden. Die Zuteilung der Punkte zu Falten erfordert eine Tiefensuche über den triangulierten Graph. Alle Dreiecke, die nicht durch Kanten des Polygonzuges getrennt sind, gehören zur selben Falte. Nachdem festgestellt wurde, dass die Dreiecke zur selben Falte gehören, werden in allen Eckpunkten dieser Dreiecke einfach alle gefundenen Fenster eingetragen. Daher muss zur Zuteilung der Falten jedes Dreieck nur einmal herangezogen werden, und die Zuteilung der Punkte zu Falten ist in $O(m)$ Zeiteinheiten erledigt.

Bei den anderen Punkten wird zur Berechnung der kürzesten Pfade der Algorithmus von Guibas et al verwendet [Gui87]. Er setzt eine Triangulierung des Polygons voraus, die mit einem Algorithmus von Tarjan und van Wyck [Tar88] in $O(m \log\log m)$ Zeiteinheiten ermittelt werden kann. Zur Berechnung der kürzesten Pfade werden Trichter verwendet, die durch das Einfügen neuer kürzester Wege geteilt werden. Eine Rekursionsformel ergibt für die Suche nach kürzesten Wegen eine Laufzeit von $O(m \log m)$. Da es $O(m)$ solcher Ebenen gibt, ergibt sich eine Laufzeit von

$$O(m^2 \log m) = O(n \log \sqrt{n}).$$

1.4.3 Beleuchtung durch Visibility Cones

Die Informationen der Visibility Cones sind nur an ihren Eckpunkten genau bestimmt worden. Dazwischen werden die Punkte durch Strecken verbunden, obwohl aufgrund der durchgeführten Berechnung dort eigentlich keine Aussage möglich ist. Die Erklärung, warum Stewart die Rendering Equation mit so ungenau bestimmten Visibility Cones lösen kann, liegt in der Art der Lichtquelle, die er auswertet. Er verwendet eine einfarbige, umgebende Lichtquelle und berechnet die Selbstabschattung des Polygonnetzes aufgrund der Visibility Cones unter Annahme eines diffusen Beleuchtungsmodells. Hierbei spielt für jeden Punkt der unverdeckte Raumwinkel seiner Umgebung eine entscheidende Rolle, während die nur ungenau bestimmten Ränder nur kleinen Einfluss auf das ausgewertete Integral haben.

Gegenüber der Rendering Gleichung (3) gelten also folgende vereinfachenden Annahmen:

- Alles Licht kommt von außen, der emittierende Term $L_e(.)$ der betrachteten Punkte $\vec{x}$ ist also 0
- Die eingestrahlte Strahldichte $L_i(.)$ ist für alle betrachteten Punkte gleich und kann daher vor das Integral gezogen werden
- Es wird ein diffuses Reflexionsmodell angenommen, die BRDF kann also durch eine Konstante wiedergegeben werden und ω_o, die Richtung des reflektierten Lichts, kann vernachlässigt werden

Es ergibt sich folgende Lösung der Rendering Gleichung:

$$L_o(\vec{x}) = L_i \cdot \int_{\vec{\omega}_i \in V(\vec{x})} k_d \cdot \cos(\vec{n}, \vec{\omega}_i) d\vec{\omega}_i, \tag{14}$$

wobei

$\vec{x}$:	betrachteter Punkt,
$\vec{\omega}_i$:	Kugelkoordinaten des einfallenden Lichts,
$L_o(\vec{x})$:	Strahldichte des von $\vec{x}$ ausgesendeten Lichts,
L_i:	Strahldichte des in $\vec{x}$ einfallenden Lichts in Richtung $\vec{\omega}_i$,
k_d:	diffuser Reflexionskoeffizient,
$\vec{n}$:	die Normale von $\vec{x}$,
$V(\vec{x})$:	die Menge der Richtungen, die im Visibility Cone von $\vec{x}$ nicht blockiert sind.

Auch für andere Arten von Lichtquellen kann man durch Visibility Cones feststellen, ob sie den Punkt beleuchten, zu dem der Visibility Cone gehört. Am einfachsten ist der Test für eine Punktlichtquelle: hier muss nur getestet werden, ob sie in einem Visibility Cone liegt. Für eine durch ein Polygon gegebene Lichtquelle könnte man das Polygon der Lichtquelle mit dem Visibility Cone schneiden, um den Anteil der Lichtquelle zu bestimmen, die den Punkt beleuchtet. Falls die Beleuchtung durch eine Environment Map gegeben ist, kann man den Algorithmus zum Rasterisieren von Dreiecken verwenden, um festzustellen, welche Pixel der Environment Map den Punkt beleuchten.

Für eine Bestimmung von Schattengrenzen von Punktlichtquellen müssten die Ränder der Visibility Maps genauer bestimmt werden, da sie darüber entscheiden, wo eine Schattengrenze verläuft. In diesem Fall könnte das Verdeckungsproblem von jedem Punkt aus gelöst werden und die Ergebnisse als Visibility Cones verwendet werden. Verschiedene Strategien zum Beschleunigen der erforderlichen Schnitt-Tests zwischen den Dreiecken sind denkbar, dazu zählen Gitter und Vereinfachung der Visibility Cone Dreiecke durch Kollabieren von inneren Kanten des Visibility Cones.

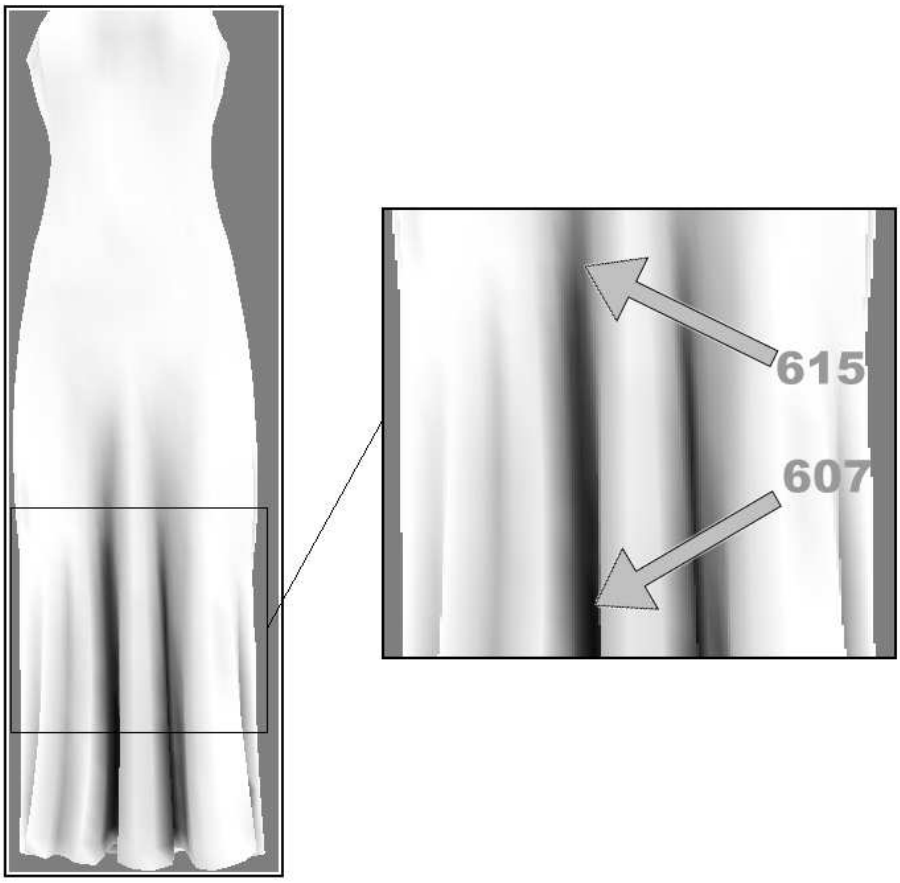

Abb.10: Zwei Punkte auf dem Mesh

2 Visibility Map Ansatz

Anstatt Visibility Cones von der CPU berechnen zu lassen, kann man auch Computergrafik Hardware einsetzen, um Bilder vom Polygonnetz aus Punkten des Polygonnetzes aufzunehmen. Diese Bilder nennen wir *Visibility Maps* im Gegensatz zu Stewarts Visibility Cones. In einer Visibility Map speichert jedes Pixel für eine Strahlrichtung, die stellvertretend für einen Raumwinkel steht, ob die entsprechende Richtung verdeckt ist oder nicht. Eine Visibility Map enthält im Vergleich zu einem Visibility Cone in allen Pixeln korrekte Verdeckungsinformation. Diese Garantie kann Stewart nur für die Eckpunkte seiner Visibility Cones geben. Im folgenden werden wir uns damit beschäftigen, Visibility Maps mit verschiedenen Algorithmen und Abbildungen zu erzeugen und sie für die Auswertung verschiedener Beleuchtungsmodelle zu benutzen.

Visibility Maps können nur für statische Szenenteile verwendet werden. Dynamische Szenenteile erfordern neue Visibility und Distance Maps bei jeder Veränderung des

Polygonnetzes. Daher sind die in dieser Arbeit dargestellten Algorithmen (noch) nicht in der Lage, in Echtzeit mit bewegter Geometrie umzugehen. Es wären Verfahren denkbar, die Visibility Maps über mehrere Frames parallel zum Rendern des Polygonnetzes aktualisieren. Wenn es gelingt, die Visibility Maps entsprechend zu priorisieren, können zunächst diejenigen Visibility Maps neu generiert werden, bei denen die größte Änderung zu erwarten ist. Dies ist aber nicht Gegenstand dieser Arbeit.

In Abb. 10 sind zwei Punkte auf dem Bild eines Polygonnetzes markiert. Ein Punkt befindet sich am Anfang einer Falte (Punkt Nummer 615 des Polygonnetzes), der andere befindet sich tief in dieser Falte (Punkt Nummer 607 des Polygonnetzes). Wir werden diese Punkte im folgenden immer wieder als Beispiele verwenden.

Bei der Erläuterung der Algorithmen sollen folgende Begriffe verwendet werden. Ein Pixel der Visibility Map gehört zum Vordergrund, wenn darin ein Stück vom Polygonnetz sichtbar ist (*Vordergrundpixel*). Sonst gehört es zum Hintergrund (*Hintergrundpixel*). Ein Vordergrund- oder Hintergrundpixel, das einen Nachbarn des entgegengesetzten Typs hat, nennen wir *Silhouettenpixel*.

Die folgenden Kapitel zeigen Algorithmen und Abbildungen zur Erzeugung von Visibility Maps auf. Anschließend werden die Möglichkeiten zur Beleuchtung mit Visibility Maps untersucht. Zum Abschluß zeigen wir die Voraussetzungen für die Anwendbarkeit der Algorithmen auf.

Um die im folgenden vorgestellten Algorithmen zu überprüfen, wurde ein Programm entwickelt, dessen Bedienung in Anhang A demonstriert wird. Dieses Programm heißt MeshShader und kann von [Gan02] heruntergeladen werden. Anhang B gibt Messergebnisse zur Laufzeit des Programms an.

2.1 Algorithmen zur Erzeugung von Visibility Maps

Bei der Erzeugung einer Visibility Map für einen Punkt muss eine Aussage über die Sichtbarkeit aller Polygone des Polygonnetzes gemacht werden. Im schlimmsten Fall müssen dazu alle Punkte des Polygonnetzes betrachtet werden. Für ein Polygonnetz aus n Punkten ergibt sich damit eine Zeitkomplexität von $O(n^2)$. Stewarts Algorithmus benötigt dagegen nur $O(n \log \sqrt{n})$ Zeiteinheiten. Dagegen sind die tatsächlichen Laufzeiten der Algorithmen schwieriger zu vergleichen, da sehr unterschiedliche Hardware benutzt wird. Obwohl die CPU-Leistung des Rechners, auf dem die Laufzeitmessungen für Anhang B vorgenommen wurden, diejenige von Stewarts damaligem Rechner um etwa den Faktor sechs übertrifft, findet ein großer Teil der Berechnungen auf der Grafikkarte statt, wodurch ein Vergleich erschwert wird. Darüber hinaus hängen die Laufzeiten zur Erzeugung der Visibility Maps von der Auflösung, dem Algorithmus und der verwendeten Abbildung ab.

Obwohl die Algorithmen in der O-Notation weit auseinander liegen, liegen ihre Laufzeiten wieder näher beieinander. Stewart berichtet für eine Jacke aus rund 3000 Punkten eine Laufzeit von ca. 20s, die bei der Erzeugung von Visibility Maps je nach verwendeter Konfiguration unterboten werden können. Wichtiger als der Vergleich der Laufzeiten sind jedoch die höhere Genauigkeit und die zusätzlichen Informationen, die Visibility Maps enthalten können, wie wir später sehen werden.

2.1.1 Triangle Rasterizer

Der Triangle Rasterizer nutzt zur Erzeugung der Visibility Maps Computergrafik Hardware über die Programmierschnittstelle OpenGL. Da die Bilderzeugung mit OpenGL zur Genüge in der Standardliteratur besprochen wird [Seg02], beschäftigen wir uns nur mit einigen Optimierungen.

Die Geometrie des Polygonnetzes wird über Triangle Strips verwaltet. Ein *Triangle Strip* ist eine Liste von Dreiecken, die so sortiert wurde, dass möglichst viele Dreiecke eine Kante mit dem vorigen Dreieck gemeinsam haben. Da man die letzte Kante des vorigen Dreiecks noch gespeichert hat, muss für das neue Dreieck nur der zusätzliche Punkt angegeben werden. Stellt man sich vor, dass die Punkte der Dreiecke in einem Schieberegister gespeichert sind, wird immer der älteste Punkt verworfen, und der neue Punkt vorne angehängt. Nach der Beschreibung des ersten Dreiecks eines Triangle Strips ergibt so jeder weitere Punkt des Triangle Strips ein weiteres Dreieck. Da Transformation und Beleuchtung für die Punkte des vorigen Dreiecks bereits durchgeführt wurden, muss dieser Schritt nur für den neuen Punkt durchgeführt werden. Dann wird die Transform and Lighting Stufe der OpenGL-Pipeline nur noch mit einem Drittel der Daten belastet. Findet man kein weiteres Dreieck mehr im Polygon-

netz, das mit dem letzten Dreieck eine Kante gemeinsam hat, muss man einen neuen Triangle Strip beginnen. Für ein Dreiecksnetz ergeben sich meist mehrere Triangle Strips, die eine sogenannte *Triangle Strip List* bilden.

Da für jeden neuen Triangle Strip die drei Punkte des ersten Dreiecks angegeben werden müssen, während für die folgenden Dreiecke jeweils der zusätzliche Punkt genügt, versucht man mit einer minimalen Anzahl von Triangle Strips auszukommen. Folgender Greedy Ansatz erlaubt eine effiziente Berechnung von Triangle Strips: Suche für jedes Dreieck der Szene nach einem Triangle Strip, an den das Dreieck angehängt werden kann. Wenn ein solcher Triangle Strip existiert, füge den neuen Punkt dem Triangle Strip hinzu. Ansonsten wird ein neuer Triangle Strip angelegt.

Im optimalen Fall liegen die Dreiecke des Netzes in einer Reihenfolge vor, bei der die Dreiecke nur an einen bestehenden Triangle Strip angehängt werden müssen. Sie lassen sich dann in linearer Zeit verarbeiten. Im schlimmsten Fall liegen die Dreiecke in einer Reihenfolge vor, bei der einzelne Dreiecke im Triangle Strip isoliert sind. Ein Dreieck nennen wir isoliert in einer Triangle Strip List, wenn es sich keinem Triangle Strip hinzufügen lässt, weil alle umliegenden Dreiecke bereits zu Triangle Strips gehören. Jedes der isolierten Dreiecke muss dann durch einen eigenen Triangle Strip verwaltet werden, und für jedes dieser Dreiecke müssen alle Triangle Strips durchsucht werden, ob das neue Dreieck an einen bestehenden Triangle Strip angehängt werden kann. Da bis zu einem Viertel der Dreiecke isoliert sein kann, ergibt sich ein Aufwand von $O(n^2)$ für die Erzeugung der Triangle Strips.

Triangle Strips lassen sich noch effizienter ermitteln, wenn man das Polygonnetz durch ein dreidimensionales *Gitter* in Zellen unterteilt. Die Zellen dieses Gitters werden als *Voxel* bezeichnet. Nachdem wir die Dreiecke des Polygonnetzes in ein Gitter einsortiert haben, wird für jede Gitterzelle eine Triangle Strip List erzeugt und als *Display List* OpenGL-intern gespeichert, sodass die Geometrie nicht durch Treiberaufrufe an OpenGL übergeben werden muss. Für einen Knoten werden dann nur die umliegenden Display Listen in einem benutzerdefinierten Radius aufgerufen. In der Praxis traten bei Begrenzung der Sichtweite auf ein Sechstel der Länge des Polygonnetzes keine sichtbaren Fehler in der Beleuchtung auf. Da nur Display Listen von Gitterzellen innerhalb der Sichtweite aufgerufen werden müssen, wird ein großer Teil der Rechenzeit eingespart (siehe Tabelle der Laufzeiten in Anhang B).

2.1.2 Raytracer

Im vorigen Abschnitt haben wir OpenGL zur Erzeugung der Visibility Maps benutzt. Die Hardware, die dabei zur Anwendung kommt, ist auf lineare Abbildungen opti-

miert. Bei Raytracing hingegen können in einfachster Weise verschiedene Abbildungen verwendet werden, da für jedes Pixel lediglich eine Strahlrichtung angegeben werden muss. Daher wurde auch ein Raytracer implementiert, um Visibility Maps zu generieren.

Bei *Appel Raytracing* [App68] wird für jedes Pixel eines zu rendernden Bildes ein Strahl in die Szene gesendet. Die verwendete Abbildung ordnet jedem Pixel eine Strahlrichtung zu. Wir testen, ob der Strahl ein Objekt schneidet. Wenn ja, tragen wir in dieses Pixel der Visibility Map ein, dass die Richtung blockiert ist. Wenn kein Objekt den Strahl schneidet, tragen wir in das Pixel der Visibility Map ein, dass die betrachtete Richtung nicht blockiert ist. Wir beschränken uns auf Polygonnetze aus Dreiecken und verwenden den Schnitt-Test für Strahlen und Dreiecke aus [Shi00].

Der Raytracer verwendet ein Gitter, um die Anzahl der erforderlichen Schnitttests zu verringern. Nachdem wir errechnet haben, in welchem Voxel der Strahl beginnt, müssen wir nur die Dreiecke in diesem Voxel gegen den Strahl testen. Wird in diesem Voxel kein vom Strahl geschnittenes Objekt gefunden, wird der Strahl solange in das nächste Voxel verfolgt, bis ein Schnittobjekt gefunden ist oder der Strahl die Szene verlässt.

Bei der Suche nach dem nächsten Voxel können aufgrund der Strahlrichtung immer drei der sechs benachbarten Voxel ausgeschlossen werden. Das Voxel, in das der Strahl als nächstes eindringt, ist durch die Voxelwand gegeben, die in Strahlrichtung als erstes vom Strahl geschnitten wird. Es gilt

$$l_x = \frac{d_x}{\Delta x}, \tag{15}$$

wobei:

- l_x: Faktor, um den der Strahl verlängert werden muss, um die nächste Voxelwand zu erreichen,
- Δx: x-Komponente der Bewegungsrichtung des Strahls,
- d_x: Distanz von der aktuellen Strahlposition zur nächsten Voxelwand.

Ähnliche Gleichungen gelten für die y- und z-Achse. Von Voxel (x, y, z) dringt der Strahl als nächstes in Voxel $(x + \text{sign}\,(\Delta x), y, z)$ falls $l_x = \min\{l_x; l_y; l_z\}$, in Voxel $(x, y + \text{sign}\,(\Delta y), z)$ falls $l_y = \min\{l_x; l_y; l_z\}$, und ansonsten in Voxel $(x, y, z + \text{sign}\,(\Delta z))$.

Die durchgeführte Berechnung liefert eine Strahlposition auf einer Voxelwand. Dies ist unschön, weil diese Position nicht eindeutig einem Voxel zugeordnet werden kann. Daher wird zusätzlich zum Eintrittspunkt in das Voxel noch der Austrittspunkt aus dem

Voxel berechnet. Der Mittelwert beider Werte dient als neue aktuelle Strahlposition.

Eine Sammlung weiterer Strategien zur Beschleunigung von Raytracern findet sich in [Enc97].

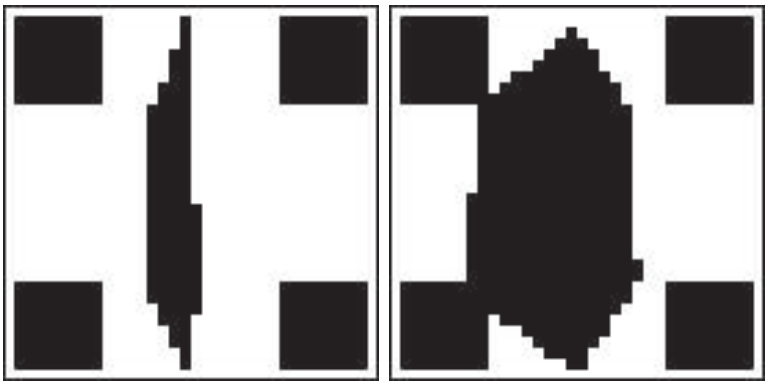

Abb.11: HemiCube Visibility Maps der Punkte aus Abb. 10 (links: Punkt 607, rechts: Punkt 615). Zur Illustration wurden die seitlichen Bilder an die entsprechenden Kanten des oberen Bildes gelegt.

2.2 Abbildungen

Wir werden uns nun mit einigen Abbildungen beschäftigen, mit denen Visibility Maps erzeugt werden können. Die Abbildung legt fest, welche Raumwinkel zu welchen Pixeln gehören. Die Abbildungen unterscheiden sich dadurch, wieviele Bilder zur Abdeckung des Halbraumes benötigt werden und ob sie mit OpenGL erzeugt werden können. Generell ist es von Vorteil, wenn eine Abbildung den Halbraum mit möglichst wenig Bildern abdeckt, denn für jedes Bild muss die komplette Geometrie verarbeitet werden. Andererseits sollte die Abbildung im homogenen Raum linear sein, damit sie auch von OpenGL durchgeführt werden kann. Dann kann der TriangleRasterizer eingesetzt werden, der schneller als der Raytracer ist. Ansonsten kann man eine Visibility Map mit nicht-linearer Abbildung auch durch Umparametrisierung einer Visibility Map erzeugen, die mit einer linearen Abbildung generiert wurde. Beim Raytracer hingegen ist es egal, ob die verwendete Abbildung linear ist oder nicht, da die Strahlrichtung in beliebiger Weise berechnet werden kann.

2.2.1 HemiCube Visibility Maps

Bei der *HemiCube*-Abbildung verwendet man die außenliegende Hälfte eines Würfels, der um den betrachteten Punkt $\vec{P}$ zentriert ist und dessen Oberseite senkrecht zur Normalen des Punktes steht, als Projektionsfläche. Der umgebende Raum wird perspektivisch auf die Seiten des Halbwürfels abgebildet, mit $\vec{P}$ als Projektionszentrum. Die Projektion wird mit der Matrix $\lim_{f\to\infty} M_{frustum}$ aus 1.3.3 durchgeführt, da später noch zusätzliche Informationen in die Visibility Map gerendert werden müssen, die sich in unendlicher Entfernung von $\vec{P}$ befinden.

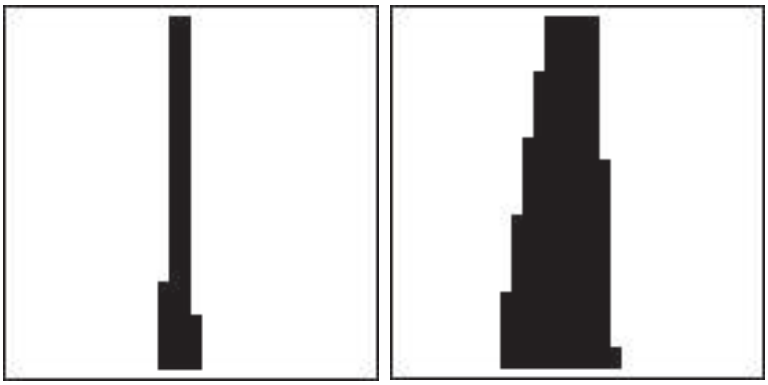

Abb.12: SinglePlane Visibility Maps der Punkte aus Abb. 10 (links: Punkt 607, rechts: Punkt 615)

Der Raytracer ordnet jedem Pixel (x, y), $x, y \in \{0; ...; s-1\}$ des oberen Bildes eine Strahlrichtung

$$\vec{d}(x,y) = \frac{2 \cdot x - s + 1}{s} \cdot \vec{x} + \frac{2 \cdot y - s + 1}{s} \cdot \vec{y} + \vec{z} \tag{16}$$

zu, wobei

s: die Auflösung der Visibility Map,
$\vec{x}$: x-Komponente der Bewegungsrichtung des Strahls, $||\vec{x}|| = 1$,
$\vec{y}$: y-Komponente der Bewegungsrichtung des Strahls, $||\vec{y}|| = 1$,
$\vec{z}$: z-Komponente der Bewegungsrichtung des Strahls, $||\vec{z}|| = 1$.

$\vec{z}$ entspricht der Normalen von $\vec{P}$. $\vec{x}$, $\vec{y}$, $\vec{z}$ definieren das lokale kartesische Koordinatensystem von $\vec{P}$. Die Kanten des umgebenden Halbwürfels sind parallel zu diesen Vektoren ausgerichtet. Die Addition von 1 auf die doppelten Pixelkoordinaten sorgt dafür, dass die Strahlen durch die Pixelmitte geschossen werden. Für die Strahlen der seitlichen Bilder kommen analoge Formeln zur Anwendung, in denen $\vec{x}$, $\vec{y}$ und $\vec{z}$ vertauscht sind. Bei den seitlichen Bildern wird die Strahlverfolgung für die untere Hälfte der Bilder nicht durchgeführt, da sich diese Strahlen auf der von der Normalen abgewandten Seite befinden.

Diese Abbildung wurde von Cohen [Coh85] für Radiosity zur Bestimmung der Formfaktoren beim Lösen des Radiosity Gleichungssystems verwendet.

2.2.2 SinglePlane Visibility Maps

Bei der SinglePlane-Abbildung beschränkt man sich auf das obere Bild des Hemi-Cubes und verbreitert lediglich den Öffnungswinkel fovy. Im TriangleRasterizer wird

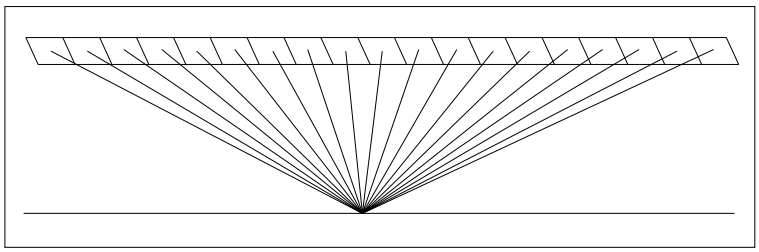

Abb.13: Problem mit der SinglePlane: die Winkel zwischen den Strahlen nehmen nach außen ab, der Bereich um die Normale wird am schwächsten abgestastet

dieser Wert als Parameter an die Funktion `glFrustum` übergeben. Beim Raytracer ändert sich die Formel für die Strahlrichtung wie folgt:

$$\vec{d}(x,y) = t \cdot \frac{2 \cdot x - s + 1}{s} \cdot \vec{x} + t \cdot \frac{2 \cdot y - s + 1}{s} \cdot \vec{y} + \vec{z}, \tag{17}$$

für $t = \tan \frac{\text{fovy}}{2}$.

Obwohl man sich für fovy einen Wert möglichst nahe an 180° wünschen würde, können meist nur wesentlich kleinere Werte verwendet werden. Weil der Bereich um die Normale eines Punktes am stärksten die Beleuchtung des Punktes beeinflußt, muss dieser Bereich auch am dichtesten abgetastet werden. Bei der SinglePlane Abbildung wird dieser Bereich aber am schwächsten abgetastet, weil der Winkel zwischen Strahlen zu benachbarten Pixeln nach außen hin auf der SinglePlane *abnimmt*, wie man in Abbildung (13) sieht. Wenn der Winkel zwischen der Normalen und dem ersten Strahl daneben zu groß wird, reicht die Abtastrate im Bereich der Normalen des Knotens nicht mehr aus, und es können Fehler in der Beleuchtung auftreten. Gleichung (17) erlaubt uns, den Winkel $\gamma = \angle(\vec{n}, \vec{d}(\frac{s}{2}, \frac{s}{2}))$ zwischen der Strahlrichtung $\vec{d}(\frac{s}{2}, \frac{s}{2})$ des ersten Pixels neben der Normalen und der Normalen $\vec{n}$ zu bestimmen. Es gilt:

$$\cos\gamma = \frac{\vec{d}(\frac{s}{2}, \frac{s}{2}) \cdot \vec{n}}{|\vec{d}(\frac{s}{2}, \frac{s}{2})| \cdot |\vec{n}|}. \tag{18}$$

Einsetzen aus Gleichung (17) ergibt:

$$\cos\gamma = \frac{1}{\sqrt{2v^2 + 1}}, \tag{19}$$

wobei

$$v = \frac{\tan \frac{\text{fovy}}{2}}{s} \tag{20}$$

fovy	120°	130°	140°	150°	160°	170°	175°
s=8	17,02	20,76	25,91	33,41	45,07	63,67	76,13
s=16	8,70	10,73	13,65	18,26	26,62	45,29	63,71
s=32	4,38	5,41	6,92	9,37	14,07	26,80	45,35
s=64	2,19	2,71	3,47	4,71	7,14	14,17	26,84
s=128	1,10	1,36	1,74	2,36	3,59	7,20	14,20
s=256	0,55	0,68	0,87	1,18	1,79	3,61	7,21

Tabelle 3: Zusammenhang zwischen α_1, fovy und s.

Abb.14: HemiSphere Visibility Maps der Punkte aus Abb. 10 (links: Punkt 607, rechts: Punkt 615)

Tabelle 3 gibt den Winkel γ zwischen der Normalen und dem ersten Strahl für verschiedene Kombinationen von Bildauflösung und fovy an. Im Test am Polygonnetz aus Abb. (10) zeigte sich, dass für $\gamma < 6^o$ brauchbare Ergebnisse für die Beleuchtung auftreten. Ab fovy $\geq 160^o$ treten auch für $\gamma < 7.5^o$ brauchbare Ergebnisse auf. Hierbei handelt es sich aber nur um eine Faustregel, die für andere Polygonnetze nicht unbedingt zutreffen muss.

2.2.3 HemiSphere Visibility Maps

Bei der HemiSphere-Abbildung wird jedem Punkt auf der Einheitskugel um den betrachteten Knoten $\vec{P}$ ein Längen- und ein Breitengrad zugeordnet. $\vec{P}$ und der durch Längen- und Breitengrad definierte Punkt auf der Kugel definieren einen Strahl, den der Raytracer durch die Szene verfolgt. HemiSphere Visibility Maps haben den Vorteil, dass der gesamte Halbraum durch ein Bild abgedeckt wird, die Abbildung ist aber auch im homogenen Raum nicht linear und kann daher nicht durch die Matrizen in OpenGL durchgeführt werden.

Der Raytracer verwendet folgende Formel zur Ermittlung der Strahlrichtung des Pixels (x, y), $x \in \{0; ...; 2s-1\}$, $y \in \{0; ...; \frac{s}{2}-1\}$:

$$\varphi(x) = \pi \cdot \frac{x+0.5}{s}, \tag{21}$$

$$\theta(y) = \pi \cdot \frac{y+0.5}{s}, \tag{22}$$

$$\begin{aligned}\vec{d}(x,y) &= \cos(\varphi(x)) \cdot \cos(\theta(y)) \cdot \vec{x} + \\ &\quad \sin(\varphi(x)) \cdot \cos(\theta(y)) \cdot \vec{y} + \sin(\theta(y)) \cdot \vec{z}, \end{aligned} \tag{23}$$

wobei

$\varphi(x)$:	Längengrad,
$\theta(y)$:	Breitengrad,
$\vec{d}(x,y)$:	Strahlrichtung für das Pixel (x,y),
s:	die Auflösung der Visibility Map,
$\vec{x}$:	x-Komponente der Bewegungsrichtung des Strahls,
$\vec{y}$:	y-Komponente der Bewegungsrichtung des Strahls,
$\vec{z}$:	z-Komponente der Bewegungsrichtung des Strahls.

Die Addition von 0.5 sorgt dafür, dass der Strahl durch die Pixelmitte geschossen wird. $\varphi(x) : \{0; ...; 2s-1\} \rightarrow (0; 2\pi)$ hat einen viermal so großen Definitions- und Wertebereich wie $\theta(y) : \{0; ...; \frac{s}{2}\} \rightarrow (0; \frac{\pi}{2})$. Dies geschieht, damit die Pixel entlang der x- und y-Achse denselben Winkel abdecken. So enthält ein Pixel entlang seiner Achsen dieselbe Informationsmenge. Da bei HemiCube / SinglePlane Visibility Maps für die x- und y-Achse derselbe Öffnungswinkel verwendet wird, wird für so erstellte Visibility Maps für beide Achsen dieselbe Anzahl von Pixeln verwendet.

Da in OpenGL keine nicht-linearen Abbildungen unterstützt werden, erzeugen wir aus einer HemiCube Visibility Map eine HemiSphere Visibility Map. Dazu bestimmen wir vorab eine Matrix, deren Einträge angeben, aus welchen Pixeln der HemiCube Visibility Map welche Pixel der HemiSphere stammen. Durch Anwendung dieser Matrix wird die Reparametrisierung zu einem einfachen Kopierdurchgang, durch den die Erzeugung einer HemiSphere Visibility Map fast so schnell geschehen kann wie die Erzeugung einer HemiCube Visibility Map.

2.2.4 Cube Visibility Maps

Oft sind Polygonnetze nicht geschlossen, d.h. nicht jede Kante eines Polygons hat ein ihr gegenüber liegendes Polygon. In diesem Fall muss für beide Seiten der Dreiecke eine getrennte Beleuchtungsrechnung durchgeführt werden. Dazu benötigt man Visibility Maps von der Ober- und Unterseite des Polygonnetzes. Wenn man den Triangle Rasterizer sowohl Ober- als auch Unterseiten der seitlichen Bilder einer HemiCube Visibility Map erzeugen lässt und zusätzlich ein Bild entgegen der Strahlrichtung aufnimmt, ergibt sich eine *Cube Visibility Map*. Sie besteht aus sechs Bildern und enthält die Verdeckungsinformation für beide Seiten einer Oberfläche. Im Vergleich zur HemiCube-Abbildung wird lediglich ein zusätzliches Bild benötigt, um Verdeckungsinformation für beide Halbräume eines Punktes zu erhalten.

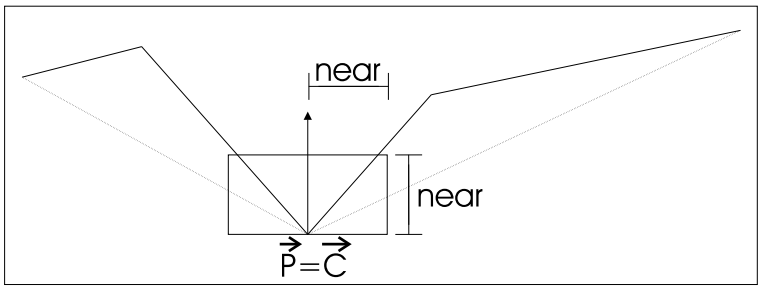

Abb.15: Wenn sich die Kamera direkt in einem Knoten befindet, wird der Bereich innerhalb der Near Clipping Plane nicht dargestellt, und die Visibility Map kann Löcher enthalten, $\vec{P}$: Knoten auf dem Polygonnetz, $\vec{C}$: Kameraposition

2.3 Probleme an der Near Clipping Plane

Setzt man die Kamera direkt in einen Knoten $\vec{P}$ des Polygonnetzes, befindet sich ein Teil der Polygone, die an $\vec{P}$ angrenzen, innerhalb der Near Clipping Plane und wird von OpenGL nicht dargestellt (Abb. 15). Der Bereich innerhalb der Near Clipping Plane wird von OpenGL nicht dargestellt. Dies kann zu Löchern in der Visibility Map führen. Da man eine Lichtquelle in eine Visibility Map transformieren kann, um zu testen, ob die entsprechende Richtung verdeckt ist, würde in diesem Fall der Algorithmus zu der Schlussfolgerung kommen, dass eine Lichtquelle in dieser Richtung nicht blockiert ist. In einer Distance Map (wird später erklärt) stellen diese Löcher zusätzliche Silhouettenpixel dar, die dort nicht auftreten dürften. Allein im Falle einer einfarbigen, umgebenden Lichtquelle ergeben sich keine wahrnehmbaren Fehler, da hier der Beleuchtungsbeitrag einzelnder Pixel keine große Rolle spielt, besonders bei den Pixeln am unteren Rand, denn diese Pixel haben nur ein sehr geringes Gewicht bei diesem Beleuchtungsmodell.

Bei einem geschlossenem Polygonnetz würde die Lücke in der Visibility Map durch ein dahinterliegendes Polygon geschlossen. Durch die Far Clipping Plane kann die Darstellung dieses Polygons aber unterdrückt werden. Der Algorithmus liefert aber auch auf nicht geschlossenen Polygonnetzen richtige Ergebnisse. Daher muss kein dahinterliegendes Polygon existieren und wir wollen einen anderen Weg suchen, diese Lücken zu verhindern. Es wäre möglich, die Visibility Map unter dem obersten Vordergrundpixel einfach zu füllen. Dies kann aber zu Fehlern führen: Zum Beispiel könnte das Polygonnetz tatsächlich Löcher haben. Ein anderes Gegenbeispiel sind Punkte am Rand einer Falte. Ein solcher Punkt kann sein ganzes Licht von der Seite erhalten, weil in der Visibility Map der Bereich um seine Normale verdeckt ist. In diesem Fall würde obiger Ansatz die Visibility Map ganz ausfüllen.

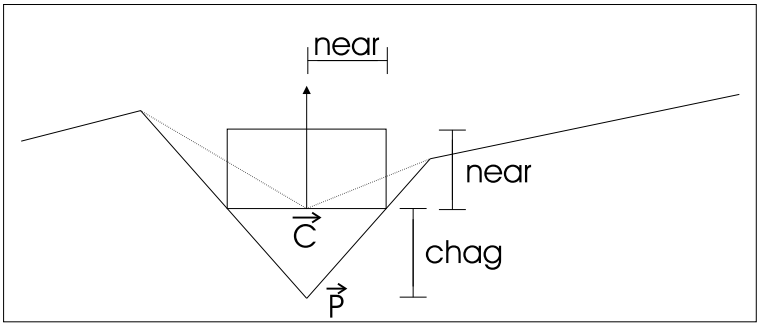

Abb.16: Behebung des Problems

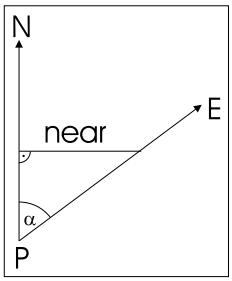

Abb.17: Herleitung einer Formel zu CHAG

Der Abstand der Near Clipping Plane vom betrachteten Punkt wird mit „near" bezeichnet. Man könnte versuchen, $\mathrm{near} = 0$ zu setzen. Dann würden aber auch die Polygone in die Visibility Map gerendert werden, die an den Punkt angrenzen. Stattdessen können wir die Kamera leicht entlang der Normalen anheben, sodass die Near Clipping Plane das Polygonnetz nicht mehr schneidet. Die Höhe, um die wir die Kamera heben, werden wir chag nennen, für Camera Height Above Ground. Da das Bild durch das Anheben der Kamera verändert wird, soll chag möglichst klein gewählt werden. Es besteht sonst die Gefahr, dass die Kamera aus einer Falte herausgehoben wird. In Abb. 16 wird die Kamera so stark angehoben, dass sich die Silhouette in der Visibility Map ändert. Normalerweise ist near aber so klein, dass ein wesentlich kleineres chag ausreicht, und daher das Bild nur unwesentlich verändert wird.

Obwohl der Raytracer ohne Near Clipping Plane arbeitet, ist auch er betroffen, da dort ein Strahl ohne chag das Netz bereits im Ursprung schneidet. Schließt man Schnittpunkte im Strahlursprung jedoch aus, können Strahlen unterhalb des Polygonnetzes verlaufen und durch ein Loch nach außen treten oder vorher gegen die Far Clipping Plane stoßen. In diesem Fall entsteht wieder ein Loch im Polygonnetz, das ebenfalls durch Anheben der Kamera vermieden werden kann.

Sei $\vec{N}$ die Normale des betrachteten Punktes $\vec{P}$ und $\vec{E}$ die Richtung der Kante mit minimalem Winkel zu $\vec{N}$. α ist der Winkel zwischen $\vec{N}$ und $\vec{E}$. Für $0 < \alpha < \pi$ kann

chag durch die im folgenden hergeleitete Formel beschrieben werden (Abb. 17):

$$\cos\alpha = \frac{\vec{N}\cdot\vec{E}}{||\vec{N}||\cdot||\vec{E}||} \tag{24}$$

$$\tan\alpha = \frac{\mathrm{near}}{\mathrm{chag}} \Rightarrow \mathrm{chag} = \frac{\mathrm{near}}{\tan\alpha} \tag{25}$$

Diese Formel gilt für diejenige der adjazenten Kanten von $\vec{P}$, die mit der Normalen des Punktes den kleinsten Winkel bildet. Falls α größer als 90° ist, können keine Löcher entstehen, da in diesem Fall der Halbraum um die Normale leer ist. Dann muss die Kamera nur um einen kleinen Wert ϵ angehoben werden, um den Fehler im Raytracer zu vermeiden. Für $\alpha > 90°$ wird nach obiger Formel chag aber negativ. Wir wählen in diesem Fall $\mathrm{chag} = \epsilon$. Ist $\alpha = 0$, so ist $\vec{E}$ parallel zu $\vec{N}$. Dann wird $\tan\alpha = 0$ und obiger Bruch ist nicht mehr definiert. Daher setzen wir in diesem Fall $\mathrm{chag} = \epsilon$. Zwar können wir in diesem Fall durch chag nicht verhindern, dass die Near Clipping Plane gegen die Bildebene stößt, aber der Fall, dass die Normale $\vec{N}$ parallel zu einer Kante $\vec{E}$ verläuft, ist ohnehin ein Fehler. Bei der Abbildung SinglePlane verbreitert sich die Bildebene um den Faktor $\tan\mathrm{fovy}$, also ergibt sich insgesamt:

$$\mathrm{chag} = \begin{cases} \frac{\mathrm{near}\cdot\tan\mathrm{fovy}}{\tan\alpha} & \text{für } 0 < \alpha < \pi, \\ \epsilon & \text{sonst.} \end{cases} \tag{26}$$

Dieser Ansatz ist nicht mit Cube Visibility Maps kompatibel, da durch Anheben der Kamera entlang der Normalen die untere Hälfte des Cubes nicht mehr in das Polygonnetz ragen würde, und somit die andere Seite des Polygonnetzes nicht berücksichtigt würde. Stattdessen müssen zwei HemiCube Visibility Maps auf verschiedenen Seiten des Polygonnetzes verwendet werden, oder eine der anderen Abbildungen, wenn eine zweiseitige Beleuchtung durchgeführt werden soll.

2.4 Punktlichtquellen und Berechnung von Schattengrenzen mit Visibility Maps

Im folgenden werden wir uns damit beschäftigen, wie Visibility Maps zur Auswertung verschiedener Beleuchtungsmodelle verwendet werden können. Dabei sind drei verschiedene Arten von Lichtquellen möglich: Punktlichtquellen, umgebende, einfarbige Lichtquellen und Beleuchtung durch Bilder einer Umgebung.

2.4.1 Visibility Maps und Punktlichtquellen

Mit Hilfe der Visibility Map eines Punktes $\vec{P}$ können wir feststellen, ob eine Punktlichtquelle $\vec{L}$ durch das Polygonnetz verdeckt ist. Dazu transformieren wir $\vec{L}$ in das Koordinatensystem der Visibility Map mittels der zum Erstellen der Visibility Map genutzten Abbildung (siehe Abschnitt 2.2). Enthält die Visibility Map an dieser Stelle ein Hintergrundpixel, ist die Lichtquelle vom Punkt aus sichtbar und erhält von diesem Licht. Wenn dort ein Teil des Polygonnetzes sichtbar ist, verdeckt dieses die Lichtquelle und der Knoten liegt im Schatten der Lichtquelle. Man kann also durch Transformation der Lichtquelle und Auswertung eines Pixels der Visibility Map entscheiden, ob der Knoten im Schatten der Lichtquelle liegt. Anschließend kann für $\vec{P}$ ein beliebiges Beleuchtungsmodell ausgewertet werden. Dabei werden ausschließlich die Lichtquellen zur Beleuchtung verwendet, bei denen die vorigen Berechnungen ergaben, dass sich $\vec{P}$ nicht in deren Schatten befindet.

Dieses Verfahren funktioniert nur für Lichtquellen, die sich außerhalb der konvexen Hülle des darzustellenden Polygonnetzes befinden, da durch ein Pixel der Visibility Map ein Raumwinkel vollständig für Lichtquellen blockiert oder freigegeben wird, und zwar unabhängig von seiner Entfernung zur Lichtquelle. Anstelle der Verdeckungsinformation der Visibility Map wäre es möglich, den Depth Buffer auszulesen und dessen Inhalt zu speichern. Man erhielte dabei eine Shadow Map aus der Perspektive eines Knotens, nicht der Lichtquelle, wie beim Shadow Map Schatten-Algorithmus. Dann könnte die Entfernung der transformierten Lichtquelle mit dem Eintrag der so entstandenen Depth Map verglichen werden. Die Beschränkung, dass sich eine Lichtquelle nur außerhalb des darzustellenden Polygonnetzes befinden darf, würde so aufgehoben.

Da bei der Abbildung SinglePlane nur ein eingeschränkter Öffnungswinkel verwendet werden kann, werden außerhalb dieses Öffnungswinkels nur Schatten angezeigt. Die Abbildung SinglePlane ist daher für die Berechnung von Schattengrenzen nicht so gut geeignet.

2.4.2 Berechnung scharfer Schattengrenzen

Wenn beim Rendering Schattengrenzen nicht besonders behandelt werden, wird an der Grenze zwischen beleuchteten und unbeleuchteten Knoten linear interpoliert. Dabei entsteht zwischen dem Kernschatten und dem beleuchteten Bereich ein weicher Übergang, wie er für einen Halbschatten typisch ist. Die Schatten einer Punktlichtquelle müssen aber scharf umrandet sein. Außerdem wird die Dreiecksstruktur des Polygonnetzes durch den Mach Band Effekt sichtbar (Abschnitt 1.1.5), und bei Animation springt der interpolierte Schatten von einem Dreieck zum nächsten (siehe Videos auf [Gan02]). Dies soll durch Berechnung und Darstellung der Schattengrenze vermieden werden. Wir beschränken uns bei der Darstellung des Algorithmus auf Polygonnetze aus Dreiecken. Nach der Berechnung wird ein Dreieck, durch das die Schattengrenze verläuft, in mehrere Dreiecke entlang der Schattengrenze unterteilt (Abb. 18). Wir führen also ein Remeshing entlang der Schattengrenze durch. Dies wird für alle Dreiecke und alle Lichtquellen wiederholt, um die Schattengrenzen aller Dreiecke festzustellen.

Wenn mehrere Lichtquellen verwendet werden, können mehrere Schattengrenzen durch ein Dreieck laufen. Dementsprechend müssen die Dreiecke dann wiederholt unterteilt werden. Um dies effizient tun zu können, wird zunächst ein Lichtquellencode für jeden Knoten der Geometrie berechnet. Der Code besteht aus einem Bit für jede Lichtquelle, das angibt, ob sich der Knoten im Schatten dieser Lichtquelle befindet. Unterscheiden sich die Codes für die Eckpunkte des Dreiecks, läuft mindestens eine Schattengrenze durch das Dreieck. Dann wird das Dreieck für jede Lichtquelle, für die der Lichtquellencode eine Schattengrenze erfordert, unterteilt. Schattendetails, die nicht auf einen Knoten des Meshes fallen, können durch die Lichtquellencodes nicht festgestellt werden, und werden vom Algorithmus daher ignoriert.

Abb. 19 zeigt die Situation bei der Berechnung der Schattengrenze zwischen zwei Punkten $\vec{P}$, $\vec{Q}$. $\vec{K}$ ist der äußerste Vorsprung eines Hindernisses, dessen Schattengren-

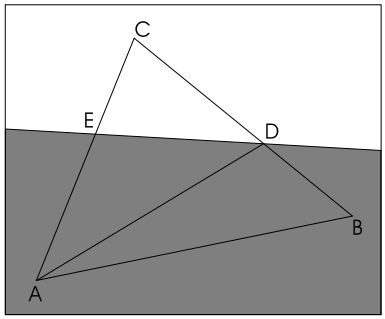

Abb.18: Dreieck ΔABC zerfällt entlang einer Schattengrenze in die Dreiecke ΔABD, ΔADE und ΔEDC

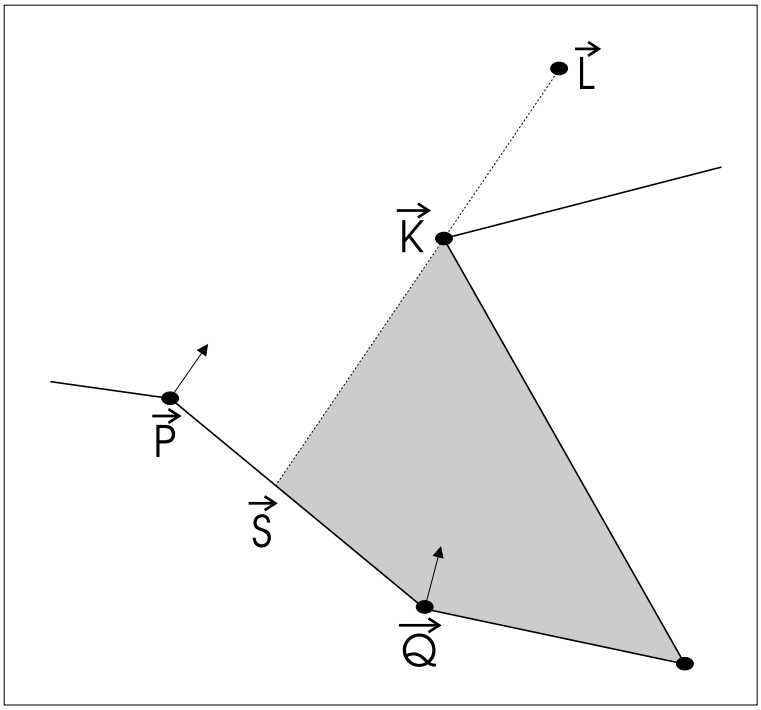

Abb.19: Berechnung der Schattengrenze zwischen P, Q

ze zwischen $\vec{P}$ und $\vec{Q}$ liegt. $\vec{L}$ ist die Lichtquelle, die diesen Schatten hervorruft. $\vec{S}$ ist die Schattengrenze zwischen $\vec{P}$, $\vec{Q}$. Da $\vec{P}$, $\vec{Q}$ und $\vec{L}$ eine Ebene bilden, in der auch $\vec{K}$ und $\vec{S}$ liegen, zeigt die Skizze nur den zweidimensionalen Fall. Gesucht wird das Teilverhältnis r, in dem $\vec{S}$ die Strecke $\overline{PQ}$ teilt. Wir werden zuerst einen zusätzlichen Map-Typus einführen, der uns bei der Berechnung von r hilft. Anschließend werden wir mit Hilfe dieser Maps r bestimmen. Zuletzt wird ein spezielles Beleuchtungsmodell für Punkte an der Schattengrenze angegeben.

2.4.3 Berechnung von Distance Maps

Wenn wir zwei Punkte $\vec{P}$, $\vec{Q}$ betrachten, von denen nur $\vec{Q}$ im Schatten einer Lichtquelle $\vec{L}$ liegt, werden wir feststellen, dass zur transformierten Position von $\vec{L}$ in der Visibility Map von $\vec{P}$ ein Hintergrundpixel gehört, während $\vec{Q}$ dort ein Vordergrundpixel hat. Also muss zwischen $\vec{P}$ und $\vec{Q}$ eine Schattengrenze verlaufen, die von einem Hindernis $\vec{K}$ verursacht wird. Wir machen die Annahme, dass der Winkel zwischen $\vec{K}$, $\vec{P}$ und $\vec{L}$ durch das nächste Silhouettenpixel gegeben ist, das von der transformierten Position der Lichtquelle $\vec{L}$ in den Visibility Maps der Punkte $\vec{P}$ und $\vec{Q}$ gefunden werden kann. Aufgrund dieser Annahme steht der Winkel zu einem Hindernis für jedes Pixel einer Visibility Map fest und diese Winkel können in einer sogenannten *Distance Map* gespeichert werden. Um den Winkel zwischen $\vec{L}$ und dem Hindernis herauszufinden,

genügt es dann, $\vec{L}$ in diese Map zu transformieren und daraus den Winkel abzulesen. Bevor wir uns jedoch weiter damit beschäftigen, wie aus zwei Winkeln die Schattengrenze $\vec{S}$ berechnet werden kann, müssen wir uns zuerst damit beschäftigen, wie eine Distance Map berechnet wird.

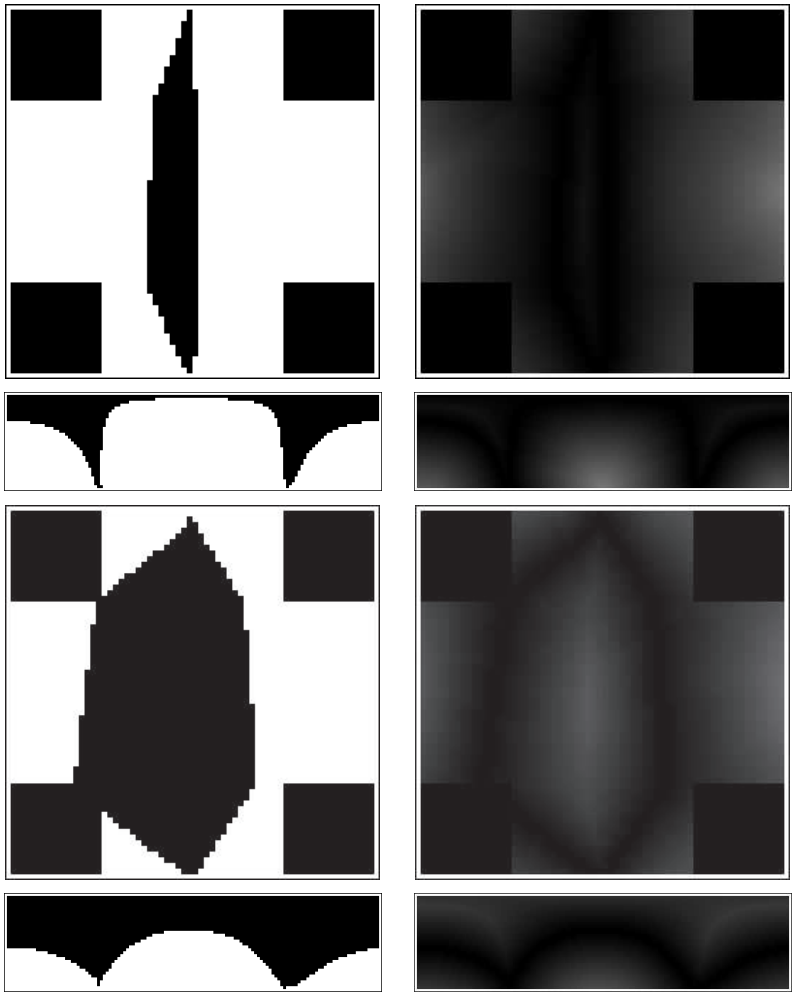

Abb.20: Visibility Maps (links) und Distance Maps (rechts) der Punkte 607 (oben) und 615 (unten) aus Abb. (10), vergleiche Abb. (11) und (14). Größere Winkel werden durch eine hellere Farbe repräsentiert.

Bei der Erzeugung der Distance Maps müssen wir besonders im Auge behalten, welche Pixel benachbart sind. Zwei Pixel einer Visibility Map heißen benachbart, wenn ihre zugehörigen Raumwinkel sich berühren. Bei der SinglePlane sind nur solche Pixel benachbart, die direkt nebeneinander liegen. Bei der HemiSphere-Abbildung sind aber auch Pixel vom linken Rand der Map mit den Pixeln auf dem rechten Rand benachbart. Außerdem sind alle Pixel am oberen Rand benachbart. Bei HemiCube sind die

Pixel über verschiedene Ränder der Bilder benachbart. Wir müssen bei der Distanzberechnung also auch Pixel berücksichtigen, die über die Ränder der Bilder benachbart sind. Es ist vorteilhaft, die Visibility Maps in einen entsprechenden Graph zu verwandeln, in dem die Pixel Knoten bilden und zwischen benachbarten Knoten Kanten einzufügen. Dies erlaubt eine uniforme Behandlung verschiedener Abbildungen bei der Erzeugung von Distance Maps aus Visibility Maps. Wir benötigen ferner dreidimensionale Koordinaten der Knoten, um zwischen den Knoten Winkel berechnen zu können. Bei HemiSphere liegen diese Koordinaten auf einer Kugel, bei HemiCube auf einem Würfel und bei SinglePlane in einem Quadrat. Alle Silhouettenpixel werden im Graph mit einer Markierung versehen.

Die Vorder- und Hintergrundpixel der Distance Map müssen mit verschiedenen Werten initialisiert werden. Für Vordergrundpixel bildet der äußere Bildrand immer eine Silhouette. Wenn γ der Winkel zwischen der Normalen von $\vec{P}$ und der Richtung ist, die das Pixel repräsentiert, dann ist $\pi - \gamma$ der Winkel zwischen der Richtung, die dem Pixel zugeordnet ist, und der Begrenzung des Halbraumes, die durch den Punkt und dessen Normale gegeben ist. Daher werden alle Vordergrundpixel mit dem Winkel $\pi - \gamma$ initialisiert. Für Hintergrundpixel bildet die äußere Bildkante hingegen keine Silhouette mehr, da die betrachtete Richtung für Licht von außen bereits blockiert ist. Daher initialisieren wir diese Pixel der Distance Map mit dem Winkel 2π. Da wir von $\vec{P}$ annehmen, dass er sich auf der Oberfläche des Polygonnetzes befindet, haben wir zusätzlich zu diesem Vordergrundpixel mindestens ein Hintergrundpixel und somit eine Silhouette. Der Winkel zu dieser Silhouette wird später den initialisierten Wert ersetzen.

Wir starten nun eine Breitensuche von den Silhouettenpixeln zur Feststellung der Winkel. Dazu verwalten wir eine Liste von Pixeln, die wir verarbeiten müssen. Diese enthält zunächst alle Silhouettenpixel. In jeder Iteration entnehmen wir der Liste ein Pixel $\vec{P}$ und überprüfen alle Nachbarn $\vec{Q}_i$, ob $\vec{P}$ ein näheres Silhouettenpixel für $\vec{Q}_i$ darstellt. In diesem Fall aktualisieren wir die Daten in $\vec{Q}_i$ und fügen $\vec{Q}_i$ der Liste hinzu. Wenn die Liste der zu verarbeitenden Pixel leer ist, wurde für jedes Pixel ein nächstes Silhouettenpixel gefunden.

Die Berechnung von Distance Maps ist ein wohlbekanntes Problem in der Informatik. Es kann durch Voronoi-Diagramme und Nearest Neighbour Maps gelöst werden. Einige graphen-basierte Algorithmen werden dazu in [Meh99] vorgestellt. Das Problem kann auch mit Hilfe von Grafik-Hardware gelöst werden [Hof99]. Dieser Ansatz wurde hier nicht gewählt, da die Distanzen über Bildränder hinweg winkeltreu propagiert werden müssen.

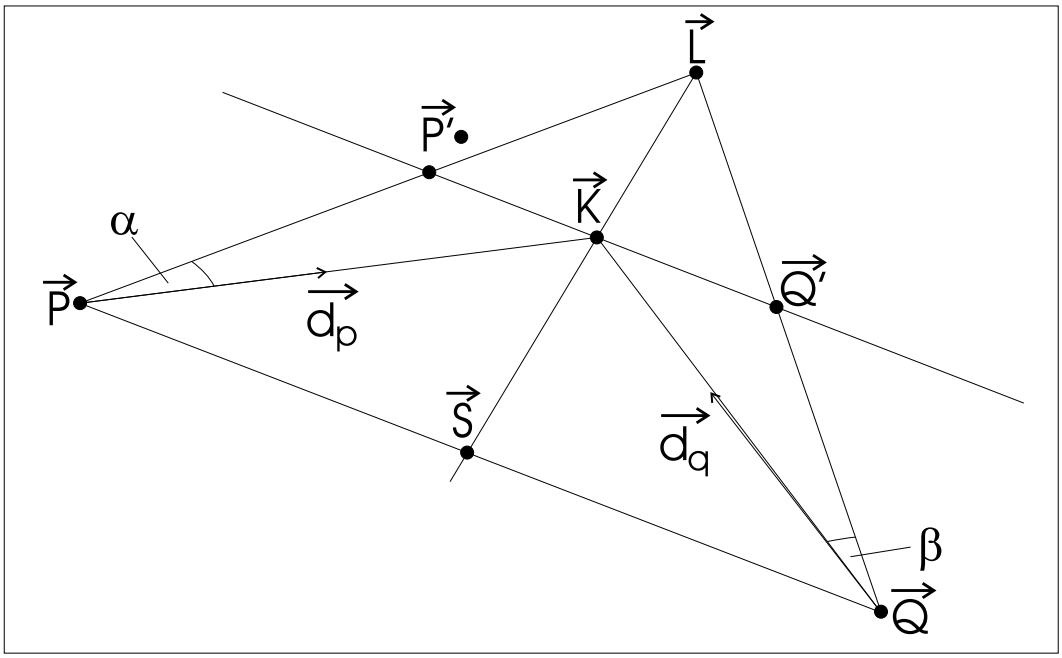

Abb.21: Berechnung von r aufgrund der Winkel α, β

2.4.4 Verwendung von Distance Maps zur Berechnung von Schattengrenzen

Nun beschäftigen wir uns mit der Berechnung des Teilverhältnisses

$$r = \frac{||\vec{P} - \vec{S}||}{||\vec{P} - \vec{Q}||}, \tag{27}$$

in dem die Schattengrenze $\vec{S}$ eine Kante zwischen zwei benachbarten Punkten $\vec{P}$ und $\vec{Q}$ teilt, mit Hilfe der der Winkel $\alpha := \angle(\vec{L}, \vec{P}, \vec{K})$ und $\beta := \angle(\vec{L}, \vec{Q}, \vec{K})$. Diese Winkel werden aus der Distance Map gelesen, wie im vorigen Abschnitt beschrieben. Abb. 21 verdeutlicht die Lage. Gerade g ist die Parallele zu $\overline{PQ}$ durch $\vec{K}$. $\vec{P}'$ ist der Schnittpunkt der Geraden $\overline{PL}$ mit g, $\vec{Q}'$ ist der Schnittpunkt der Geraden $\overline{QL}$ mit g.

Wir drehen den Richtungsvektor $\vec{L} - \vec{P}$ um den Winkel α in Richtung des Winkels $\vec{Q} - \vec{P}$, und erhalten einen Vektor d_p. Analog drehen wir den Vektor $\vec{L} - \vec{P}$ um den Winkel β in Richtung des Winkels $\vec{P} - \vec{Q}$ und erhalten einen Vektor d_q. d_p und d_q sind die Richtungsvektoren der Geraden durch $\overline{PK}$ und $\overline{QK}$, also können wir $\vec{K}$ aus deren Schnittpunkt errechnen. Nun läßt sich $\vec{S}$ aus dem Schnittpunkt der Geraden durch $\overline{PQ}$ und $\overline{LK}$ bestimmen, und somit auch unser Teilverhältnis r.

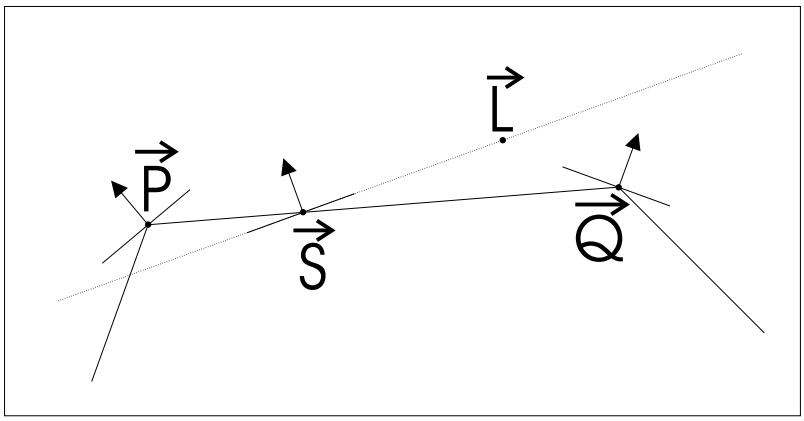

Abb.22: Grenze zur Selbstabschattung

2.4.5 Näherung für kleine α, β

Für kleine Winkel α, β kann die Schattengrenze zwischen zwei Punkten $\vec{P}$, $\vec{Q}$ auch anders bestimmt werden. In diesem Fall gilt

$$\frac{||\overline{P'K}||}{\alpha} \approx \frac{||\overline{Q'K}||}{\beta}. \tag{28}$$

Es muss also ein c existieren, sodass $||\overline{P'K}|| = c \cdot \alpha$ und $||\overline{Q'K}|| = c \cdot \beta$ gelten. Dann folgt

$$r = \frac{||\overline{PS}||}{||\overline{PQ}||} = \frac{||\overline{P'K}||}{||\overline{P'Q'}||} = \frac{||\overline{P'K}||}{||\overline{P'K}|| + ||\overline{Q'K}||} = \frac{c \cdot \alpha}{c \cdot \alpha + c \cdot \beta} = \frac{\alpha}{\alpha + \beta}. \tag{29}$$

Für kleine Winkel α, β lässt sich r also sehr einfach bestimmen.

2.4.6 Schattengrenze für Punkte, deren Normale von der Lichtquelle abgewandt ist

Es ist möglich, dass sich eine Lichtquelle außerhalb des der Normalen zugewandten Halbraumes eines Punkts befindet. Dann kann der Winkel zum verdeckenden Hindernis nicht der Distance Map des oberen Halbraums entnommen werden, da dieser Bereich von der Visibility Map nicht abgedeckt wird. Ein Lösungsansatz besteht in der Anwendung der im folgenden hergeleiteten Formel für die Berechnung der Schattengrenze. Abb. 22 stellt die Situation dar. Da sich $\vec{L}$ im Halbraum von $\vec{Q}$ befindet, nicht aber im Halbraum von $\vec{P}$, erhält $\vec{P}$ kein Licht von $\vec{L}$. Es muss sich eine Schattengrenze $\vec{S}$ zwischen $\vec{P}$ und $\vec{Q}$ befinden. Diese zeichnet sich dadurch aus, dass der Richtungsvektor von $\vec{S}$ zu $\vec{L}$ genau senkrecht auf die Normale von $\vec{S}$ steht, denn $\vec{L}$

liegt für diesen Punkt genau auf der Ebene, die die Halbräume von $\vec{S}$ trennt. Folgende Gleichungen müssen erfüllt sein:

$$\vec{S} = r \cdot \vec{P} + (1 - r) \cdot \vec{Q} \tag{30}$$

$$\vec{N}_S = r \cdot \vec{N}_p + (1 - r) \cdot \vec{N}_Q \tag{31}$$

$$\vec{N}_S \cdot (\vec{S} - \vec{L}) = 0 \tag{32}$$

Hierbei ist mit $\vec{N}_X$ wie immer die Normale von $\vec{X}$ gemeint. Beim Einsetzen von 30, 31 in 32 erhält man:

$$\left[r \cdot \vec{N}_p + (1 - r) \cdot \vec{N}_Q\right] \cdot \left[r \cdot \vec{P} + (1 - r) \cdot \vec{Q} - \vec{L}\right] = 0. \tag{33}$$

Bei der Lösung dieses quadratischen Gleichungssystems in drei Gleichungen ergeben sich zwei Lösungen, von denen durch Probe eine ausgewählt wird. Die exakte Lösung würde (33) erfüllen, aber aufgrund der begrenzten Genauigkeit der Gleitkomma-Arithmetik können wir nicht davon ausgehen, dass eine der Lösungen die Gleichung exakt erfüllt. Wir wählen daher die Lösung, die $\vec{N}_S^0 \cdot (\vec{S} - \vec{L})$ minimiert, $\vec{N}_S^0 = \frac{\vec{N}_S}{||\vec{N}_S||}$. Es ist wichtig, $\vec{N}_S^0$ anstelle von $\vec{N}_S$ zu verwenden, um den Abstand von $\vec{L}$ von der durch $\vec{S}$ und $\vec{N}_S$ definierten Ebene richtig berechnen zu können.

Eine Alternative bestünde darin, für den der Normalen abgewandten Halbraum eine zusätzliche Distance Map zu bestimmen. Die Auswahl der richtigen Distance Map benötigt dann einen zusätzlichen Halbraumtest. Mit diesem wird festgestellt, ob die obere oder untere Distance Map eines Punkts zur Anwendung kommt. Alle Pixel der unteren Distance Map sind automatisch Hintergrundpixel; sie müssen an den entsprechenden Rändern mit den oberen Pixeln im Graph verbunden werden. Aus der unteren Distance Map ließen sich dann ganz normal die Winkel zum nächsten verdeckenden Hindernis auslesen.

2.4.7 Beleuchtung an der Schattengrenze

Aus dem Teilverhältnis r lässt sich nicht nur $\vec{S}$ berechnen, sondern es erlaubt auch die Interpolation anderer Eigenschaften der Dreiecke:

$$\vec{S} = r \cdot \vec{P} + (1 - r) \cdot \vec{Q} \tag{34}$$

$$\vec{N}_S = r \cdot \vec{N}_P + (1 - r) \cdot \vec{N}_Q \tag{35}$$

$$\vec{N}_S^0 = \frac{\vec{N}_S}{||\vec{N}_S||} \tag{36}$$

$$L_{S,amb} = r \cdot L_{P,amb} + (1 - r)L_{Q,amb} \tag{37}$$

$$L_{S,diff} = r \cdot L_{P,diff} + (1 - r)L_{Q,diff} \tag{38}$$

wobei

$\vec{S}$:	die Schattengrenze ist,
r:	das Teilverhältnis, in dem s die Strecke $\overline{P}Q$ teilt,
$\vec{N}_X$:	die Normale von $\vec{X}$,
$\vec{N}_S^0$:	die normierte Normale von $\vec{S}$,
$L_{X,amb}$	die ambiente Strahldichte von Punkt $\vec{X}$,
$L_{X,diff}$	die diffuse Strahldichte von Punkt $\vec{X}$.

Beim neuen Punkt werden Verweise auf die umliegenden Punkte gespeichert, um für weitere Lichtquellen die neu entstandenen Dreiecke weiter unterteilen zu können. Obwohl alle erforderlichen Informationen vorhanden sind, sollte das Beleuchtungsmodell nicht an der Schattengrenze ausgewertet werden. Während diesen Pixeln vorher eine interpolierte Strahldichte zugeordnet wurde, wird ihnen plötzlich der korrekt berechnete Wert zugewiesen. Dies ist bei einer Animation ein sichtbarer, störender Effekt, weil die neu berechnete Strahldichte nicht zur interpolierten Strahldichte passt, die dem Pixel zuvor zugeordnet war. Dieser Effekt wird verhindert, indem die Strahldichte auf beiden Seiten der Schattengrenze interpoliert wird. Auf der beleuchteten Seite wird zwischen den Strahldichten interpoliert, die sich für die eingeschaltete Lichtquelle ergeben. Auf der anderen Seite wird zwischen den Strahldichten interpoliert, die sich für die abgeschaltete Lichtquellen ergeben. Dadurch wird die Gouraud-Interpolation, die OpenGL verwendet, genau nachgebildet.

Dies erfordert, dass an den unterteilten Kanten nicht ein Punkt, sondern zwei Punkte eingefügt werden. Der eine Punkt wird von der Lichtquelle beleuchtet, der andere nicht. Die Position der Punkte ist identisch, aber sie unterscheiden sich im Lichtquellencode. Für die Eckpunkte des Dreiecks werden die Lichtbeiträge der einzelnen Lichtquellen getrennt gespeichert, um durch einen Lichtquellencode das Beleuchtungsmodell auswerten zu können.

Wir verwenden folgendes Verfahren zur wiederholten Unterteilung der Dreiecke durch mehrere Schattengrenzen: Bei einem durch Unterteilung neu entstandenen Punkt wurden Verweise auf die umliegenden Punkte gespeichert. Um für einen solchen Punkt eine weitere Schattengrenze zu ermitteln, transformiert man die Lichtquellen in die drei Distance Maps der umliegenden Punkte und gewichtet deren Werte nach dem Abstand der Punkte zum betrachteten Punkt, zu denen die Distance Maps gehören. So lässt sich durch trilineare Interpolation für eine Lichtquelle aus den drei Maps der Winkel α bzw. β berechnen, ohne eine komplette Distance Map zu berechnen zu müssen. Dies setzt voraus, dass Distance Maps vorzeichenbehaftet gespeichert werden. Das Vorzeichen

codiert, ob es sich um ein Vordergrund- oder Hintergrundpixel handelt.

2.4.8 Schlußfolgerungen

Die Laufzeitmessungen in Anhang B belegen, dass der Algorithmus echtzeitfähig ist. Leider ist die Rechengenauigkeit durch die Auflösung der Distance Maps begrenzt. Oft liegt eine Lichtquelle für den einen Punkt nur knapp unter der Silhouette, für den anderen Punkt nur knapp darüber. In diesem Fall entscheiden wenige Pixel über die Winkel α und β, und dies begründet den Bedarf an Distance Maps hoher Auflösung. Für die Erzeugung der Distance Maps werden z.B. HemiSphere Visibility Maps benötigt, die eine Auflösung von 128×32 oder 256×64 haben. Da diese Maps für jeden Knoten des Polygonnetzes benötigt werden, sind die Speicheranforderungen beachtlich. Auch die Rechenzeit für die Ermittlung der Distance Maps ist hoch.

Unter [Gan02] findet sich ein Video zum vorgestellten Algorithmus.

2.5 Beleuchtung von Polygonnetzen mit Visibility Maps und High Dynamic Range Environment Maps

Ziel dieses Abschnitts ist es, Möglichkeiten zur Nachahmung der Beleuchtungssituation an einem beliebigen Ort aufzuzeigen. Da die Beleuchtung durch eine Cube Environment Map erfolgt, werden sechs Bilder dieses Orts für die Richtungen oben, unten, Norden, Osten, Süden und Westen benötigt. Wir gehen davon aus, dass die sechs Bilder für die verschiedenen Seiten der Cube Map in einem High Dynamic Range Format vorliegen (Abschnitt 1.2.1). Für die Beleuchtung leiten wir eine Lösung der Rendering Gleichung her, und wir müssen uns noch einmal mit den Abbildungen beschäftigen, mit denen wir die Visibility Maps erzeugen.

Wir gehen von der Rendering Gleichung aus (Formel 3), und setzen

$$d\vec{\omega}_i = \frac{\cos\theta_o}{|x - x'|^2} dA, \tag{39}$$

wobei
A: ein infinitesimal kleines Flächenelement,
θ_o: der Winkel zwischen der Normalen von A und $\vec{\omega}_o$.

Wir erhalten

$$L_o(\vec{x}, \vec{\omega}_o) = L_e(\vec{x}, \vec{\omega}_o) + \int_A \rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) \cdot L_i(\vec{x}, \vec{\omega}_i) \cdot \frac{\cos\theta_i \cos\theta_o}{|\vec{x} - \vec{x}'|^2} dA, \tag{40}$$

wobei
θ_i: Winkel zwischen der Normalen von $\vec{x}$ und $\vec{\omega}_i$.

Die Herleitung dieser Formel wurde [Enc97] entnommen. Dieses Integral werten wir nur an den Pixeln der Visibility Map aus, daher geht es in eine Summe über:

$$L_o(\vec{x}, \vec{\omega}_o) = L_e(\vec{x}, \vec{\omega}_o) + \sum_{p \in P} \rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) \cdot \frac{\cos\theta_i \cos\theta_o}{|\vec{x} - \vec{x}'|^2} \cdot A(p) \cdot L_i(p). \tag{41}$$

wobei
P: die Menge der Pixel,
p: das betrachtete Pixel,
$L_i(p)$ die dem Pixel zugeordnete Lichtquelle,
$A(p)$ Fläche des Pixels.

Wir ordnen jedem Pixel einen Geometrieterm

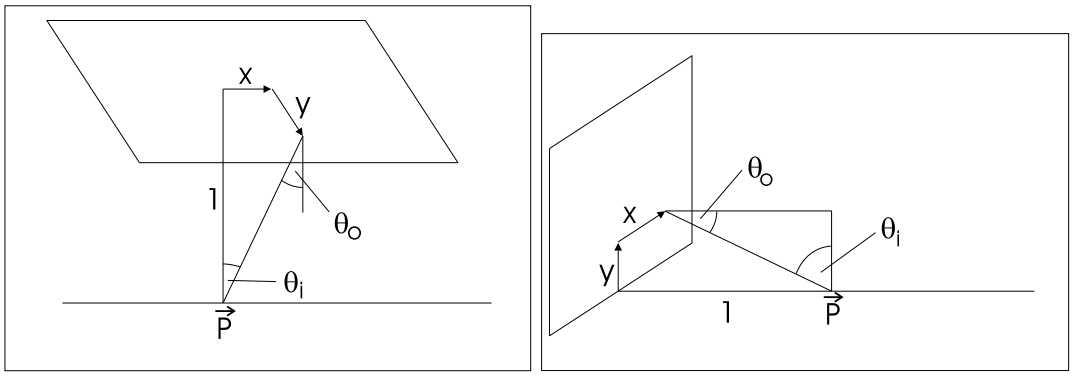

Abb.23: Oberes (links) und seitliches Bild (rechts) eines HemiCubes

$$g(p) = \frac{\cos\theta_i(p)\cos\theta_o(p)}{r(p)^2} \cdot A(p) \tag{42}$$

zu, wobei

$\theta_i(p)$: Funktion, die Pixel p den Winkel θ_i zuordnet,
$\theta_o(p)$: Funktion, die Pixel p den Winkel θ_o zuordnet,
$A(p)$: Funktion, die Pixel p eine Fläche zuordnet,
$r(p)$: Funktion, die den Abstand der Fläche $A(p)$ von $\vec{x}$ bezeichnet.

Dieser Geometrieterm entspricht den Delta-Formfaktoren aus der Radiosity [Coh85]. Wir erhalten:

$$L_o(\vec{x}, \vec{\omega}_o) = L_e(\vec{x}, \vec{\omega}_o) + \sum_{p \in P} \rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) \cdot g(p) \cdot L_i(p). \tag{43}$$

Da sich bei veränderter Betrachterposition die BRDF ändern kann, muss bei jeder Bewegung der Kamera eine neue Lösung dieser Gleichung berechnet werden.

2.5.1 HemiCube

Wir bestimmen nun den Geometrieterm für die verschiedenen Abbildungen. Den Geometrieterm für das obere Bild bezeichnen wir mit $g_s^{\mathrm{HCtop}}(x, y)$, für die seitliche Bilder mit $g_s^{HCside}(x, y)$. s gibt die Auflösung der auszuwertenden Visibility Map an. Das Koordinatenpaar $(x, y) = p$ gibt die Position des betrachteten Pixels $p \in P$ in der Visibility Map an. Abb. 23 verdeutlicht die Situation für obere und seitliche Bilder einer HemiCube Visibility Map.

Im Falle des oberen HemiCube Bildes gilt nach [Enc97] $\theta_i(p) = \theta_o(p) =: \theta(p)$, $\cos\theta(p) = \frac{1}{r(p)}$, $r(x,y) = \sqrt{x^2+y^2+1}$. Wir erhalten also

$$g_s^{\mathrm{HCtop}}(x,y) = \frac{1}{r(p)^4} \cdot \Delta A = \frac{1}{(x^2+y^2+1)^2} \cdot \Delta A, \tag{44}$$

für ΔA: die Größe der Pixel (alle Pixel sind bei HemiCube gleich groß).

Bei den seitlichen Bildern (siehe Abb. 23, rechts) gilt wieder $r(p) = \sqrt{x^2+y^2+1}$, aber diesmal $\cos\theta_i(p) = \frac{y}{r}$, $\cos\theta_o(p) = \frac{1}{r}$. Es folgt

$$g_s^{\mathrm{HCside}}(x,y) = \frac{y}{r(p)^4} \cdot \Delta A = \frac{y}{(x^2+y^2+1)^2} \cdot \Delta A. \tag{45}$$

2.5.2 SinglePlane

Bei der SinglePlane wird derselbe Geometrieterm verwendet wie beim oberen HemiCube-Bild. Die Aufweitung durch den Öffnungswinkel fovy muss jedoch als Faktor $\tan\frac{\mathrm{FOVY}}{2}$ berücksichtigt werden:

$$g_s^{\mathrm{SP}}(x,y) = \frac{1}{(x'^2+y'^2+1)^2} \cdot \Delta A, \tag{46}$$

mit $x' = x \cdot \tan\frac{\mathrm{FOVY}}{2}$, $y' = y \cdot \tan\frac{\mathrm{FOVY}}{2}$.

2.5.3 HemiSphere

Für die HemiSphere-Abbildung bezeichnen wir den Geometrieterm für Pixel $p = (x,y)$ mit $g_s^{\mathrm{HS}}(x,y)$. Es gilt $r(p) = 1$, $\theta_o(p) = 0$, $\cos\theta_o(p) = 1$. Wir erhalten:

$$g_s^{\mathrm{HS}}(x,y) = \Delta A \cdot \cos\theta_i(y). \tag{47}$$

Unter Ausnutzung von Gleichung (22) ergibt sich:

$$g_s^{\mathrm{HS}}(x,y) = \Delta A \cdot \cos\left(\pi \cdot \frac{y+0.5}{s}\right). \tag{48}$$

Nun muss noch die Fläche ΔA bestimmt werden. Es gibt $2s$ Breitenkreise mit derselben y-Koordinate. Jeder davon hat einen Anteil von $\frac{1}{2s}$ am Breitenkreis mit Umfang $2\pi\sin\theta$ für Breite θ. Durch Integration über diese Breitenkreise erhalten wir ΔA:

$$
\begin{aligned}
\Delta A &= \int_{\theta_o}^{\theta_u} \frac{4\pi}{s} \sin\theta d\theta \\
&= \frac{\pi}{s} \int_{\theta_o}^{\theta_u} \sin\theta d\theta \\
&= \frac{\pi}{s} \left[-\cos\theta\right]_{\theta_o}^{\theta_u} \\
&= \frac{\pi}{s} \cdot (\cos\theta_o - \cos\theta_u) \qquad (49)
\end{aligned}
$$

zu, wobei

$\theta_o = \pi \cdot \frac{y}{s}$: Breitengrad des oberen Randes von ΔA,

$\theta_u = \pi \cdot \frac{y+1}{s}$: Breitengrad des unteren Randes von ΔA.

2.5.4 Bildbasierte Beleuchtung mit Selbstabschattung

Bei einer normalen Environment Map wird die Environment Map als Spiegelung auf ein Objekt aufgetragen (Kapitel 1.2.2). Dabei wird für jedes Pixel des Objekts ein Pixel der Environment Map ausgewertet. Wir wollen aber jedes Pixel der Environment Map als *Richtungslichtquelle* für jeden Knoten des Objekts auffassen. In Verbindung mit einer BRDF erlaubt dies eine realistische Simulation der Beleuchtung des Ortes, von dem die Environment Map stammt.

Wir wollen also für jedes unverdeckte Pixel einer Visibility Map einen Strahl mit der Environment Map schneiden. Um die Schnittberechnung zu vermeiden, könnte man die Environment Map direkt in die Visibility Maps rendern. Da Interreflexionen vernachlässigt werden, wird das Polygonnetz in die Visibility Map in schwarz gerendert. Nun würde über die Pixel der Visibility Map nach Gleichung (43) summiert. Diese Lösung würde aber neue Visibility Maps erfordern, sobald die Environment Map sich verändern würde, z.B. bei einer Drehung des Polygonnetzes gegenüber der Environment Map.

Daher rendern wir nicht die Environment Map in die Visibility Map, sondern eine Index-Textur. Die Index-Textur codiert die Richtungen als Farben, und jeder Eintrag dient als Index in die Environment Map. In der Index-Textur wird eine Farbe für die Bildnummer der Environment Map und je eine weitere für je die x- und y-Koordinate verwendet. Da die Visibility Map nun nur noch Indizes in die Environment Map enthält, können wir die Environment Map austauschen ohne neue Visibility Maps zu generieren. Durch dieses Verfahren wird jedem Hintergrundpixel p einer Visibility Map eine Lichtquelle $L_i(p)$ zugeordnet (vergleiche Gleichung (43)). Im Raytracer

wird hingegen eine Schnittberechnung mit der Cube Environment Map durchgeführt. In der Kombination TriangleRasterizer / HemiSphere wird die Index Map aus der reparametrisierten HemiCube Visibility Map übernommen.

Das Programm MeshShader erlaubt derzeit nur die Auswertung eines diffusen Beleuchtungsmodells für Environment Maps. Zur Demonstration der Austauschbarkeit der Environment Map wurde ein Film generiert, in dem vier farbige Wände rotieren. Zwar könnte pro Frame auch eine rotierte Ansicht der Wände generiert werden, aber der Ansatz über die Index Textur pro Frame auf ein Bild eines Films zuzugreifen, ist flexibler. Anstelle eines Films könnten auch gerenderte Bilder einer beliebigen Szene verwendet werden, durch den Film wird aber vom Zeitaufwand der Bildgenerierung abstrahiert.

Zu diesem Algorithmus findet sich unter [Gan02] ein Video.

2.5.5 Beleuchtung mit einer einfarbigen, umgebenden Lichtquelle

Im Falle einer einfarbigen, umgebenden Lichtquelle und unter Annahme eines diffusen Reflexionsmodells ergibt sich ein Beleuchtungsverfahren, das Selbstabschattung berücksichtigt. Wir gehen davon aus, dass Licht nur von außen am Punkt $\vec{x}$ ankommt und können den Faktor $L_i(.)$ aus der Summe in Gleichung (43) herausziehen:

$$L_o(\vec{x}, \vec{\omega}_o) = L_i(\vec{x}, \vec{\omega}_i) \cdot \sum_{p \in P} \rho(\vec{x}, \vec{\omega}_i, \vec{\omega}_o) \cdot g(p) \cdot h(p). \tag{50}$$

Hier ist $h(p)$ wie folgt definiert:

$$h(p) = \begin{cases} 0, & \text{falls } p \text{ in der Visibility Map ein Vordergrundpixel ist,} \\ 1, & \text{sonst.} \end{cases} \tag{51}$$

In diesem Fall kann man die Summe für alle Punkte $\vec{x}$ bestimmen und zwischenspeichern. Zur Auswertung der Beleuchtung für eine einfarbige, umgebende Lichtquelle muss diese Summe nur noch mit $L_i(\vec{x}, \vec{\omega}_i)$ multipliziert werden. Dieses Beleuchtungsmodell ist invariant unter der Betrachterposition und kann äußerst schnell ausgewertet werden. Stewart wertet für Visibility Cones ein ähnliches Integral aus (Gleichung (14)).

Auch zu diesem Algorithmus findet sich unter [Gan02] ein Video.

2.6 Probleme und Restriktionen der Algorithmen

Um die vorgestellten Algorithmen für ein Polygonnetz anwenden zu können, sollten folgende Bedingungen erfüllt sein:

- Polygone dürfen sich nicht schneiden
  Wenn sich Polygone des Datensatzes schneiden, liegen Polygone teilweise im Inneren des Polygonnetzes, teilweise außerhalb. Innere Knoten des Polygonnetzes erhalten kein Licht von außen, sind also schwarz, während äußere Knoten eine hellere Farbe erhalten. Die Interpolation zwischen diesen Farbwerten führt zu Verfärbungen an den Schnittkanten, die den optischen Eindruck stören (siehe Abbildung 24).

- Nicht geschlossene Polygonnetze
  Bei nicht geschlossenen Polygonnetzen können Innen- und Außenseite des Polygonnetzes nicht unterschieden werden. Außerdem müsste die Beleuchtung für beide Seiten des Dreiecks getrennt durchgeführt werden. Dazu würden aber auch von der Innenseite des Polygonnetzes Visibility Maps benötigt.

- Der Datensatz sollte keine Löcher oder Spalten enthalten
  Am Rande einer Spalte sind die beiden benachbarten Punkte unbekannt. Daher können die Normalen der angrenzenden Flächen nicht gemittelt werden, um für $\vec{P}$ eine Normale zu berechnen. Interpolation kann dann auch nicht funktionieren, sodass Unstetigkeiten in der Beleuchtung auftreten, selbst wenn die Spalte so schmal ist, dass sie beim Rendering nicht sichtbar ist.

- Mehrere Kopien desselben Punktes
  Dies hat dieselben Auswirkungen wie Spalten im Polygonnetz, da für einen Knoten nicht identifiziert werden kann, welche Polygone zu ihm gehören.

- Innere Polygone
  Polygone im Inneren des Polygonnetzes führen nicht zu Fehlern, benötigen aber Speicherplatz und Rechenzeit. Polygone im Inneren des Datennetzes erhalten kein Licht von Außen, da äußere Lichtquellen blockiert sind. Sie sind also immer schwarz und nur sichtbar, wenn sich die Kamera im Inneren des Datennetzes befindet. Der Aufenthalt der Kamera im Inneren des Polygonnetzes würde aber an sich schon einen Fehler darstellen.

- Punkte eines Polygons nicht entgegen dem Uhrzeigersinn aufgelistet
  Zur Berechnung der Normalen des Polygons verwendet der Algorithmus die Annahme, dass die Punkte eines Polygons bei Betrachtung der Außenseite entgegen dem Uhrzeigersinn aufgelistet sind.

- Polygonnetz besteht nicht aus Dreiecken
  Unser Algorithmus behandelt nur Dreiecke.

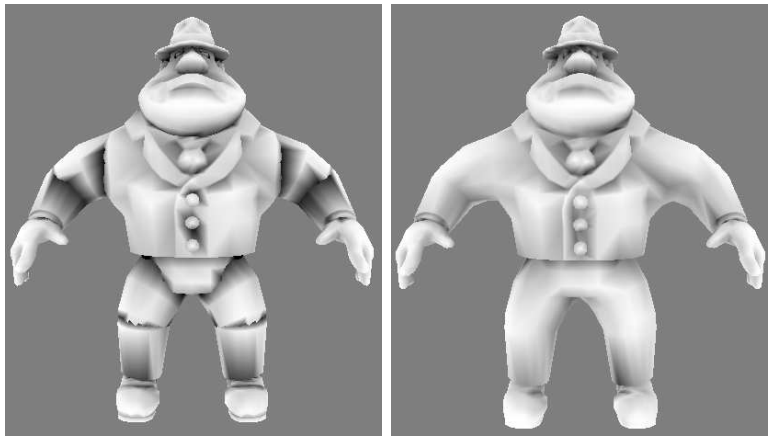

Abb.24: Der linke Teil der Abb. zeigt ein Polygonnetz aus dem Internet. Das Polygonnetz enthält eine Reihe sich schneidender Dreiecke, die zu Fehlern bei der Beleuchtung führen. Rechts sieht man eine manuell bereinigte Version des Datensatzes, die mit einer einfarbigen, umgebenden Lichtquelle beleuchtet wurde.

- Punkte, die zu keinem Dreieck gehören
  Solche Punkte belegen Speicherplatz und benötigen Rechenzeit, dennoch werden die berechneten Helligkeiten für diese Punkte nicht angezeigt.

3 Ergebnisse und Ausblick

Zum Abschluss wird kurz auf mögliche zukünftige Erweiterungen eingegangen und danach eine Schlussfolgerung gezogen.

3.1 Schattengrenzen bei Verwendung von Environment Maps

Die bisher verwendete Technik zur Auswertung von Environment Maps produziert keine Schattengrenzen zwischen den Knoten des Polygonnetzes. Außerdem wird für alle Punkte des Polygonnetzes eine Faltung zwischen den Visibility Maps aller Punkte und der Environment Map durchgeführt, egal ob diese Punkte im gerenderten Bild sichtbar sind oder nicht.

Es wäre vorteilhaft, zur Berechnung von Schattengrenzen Dreiecke dann unterteilen zu können, wenn sie im Bild eine große Anzahl von Pixeln belegen. Für Dreiecke, die in einem Bild nicht sichtbar sind, sollte die Berechnung eines Farbwertes unterdrückt werden können. Beide Ziele können erreicht werden, indem man ein Bild aus der Betrachterposition generiert, in dem jedes Dreieck in einer bestimmten Farbe dargestellt wird. Anhand dieses Bildes kann ermittelt werden, wieviele Pixel ein Dreieck im Bild belegt. Wenn ein Dreieck mehr als eine vom Benutzer einstellbare Anzahl von Pixeln auf dem Bildschirm belegt oder seine Eckpunkte einen zu großen Helligkeitsunterschied aufweisen, unterteilen wir die Kanten des Dreiecks und erhalten vier neue Dreiecke. Zu jedem Dreieck gehört dann nur noch ein Viertel der Pixel des ursprünglichen Dreiecks. Diese Unterteilung wird bei Bedarf wiederholt, bis das jeweilige Dreieck eine vom Benutzer einstellbare Anzahl von Pixeln unterschreitet. So kann der Benutzer erreichen, dass Dreiecke gar nicht unterteilt werden, oder sie solange unterteilt werden, bis jedes Dreieck im darzustellenden Bild die Größe eines Pixels hat.

An einem neu entstandenen Knoten $\vec{Q}$ kann durch trilineare Interpolation zwischen den Einträgen der Distance Maps der umliegenden Punkte ein Distance Map Eintrag $d(p)$ für ein Pixel p der Distance Map von $\vec{Q}$ errechnet werden. Wenn Vorder- und Hintergrundpixel in der Distance Map durch verschiedene Vorzeichen unterschieden werden, kann durch Auswertung des Vorzeichens von d entschieden werden, ob das Pixel p der Visibility Map ein Vorder- oder Hintergrundpixel ist. Die entstandene Visibility Map wird nicht gespeichert, da sie nur einmal zur Auswertung der Environment Map verwendet wird. Abgesehen davon wird in $\vec{Q}$ ganz normal nach Gleichung (43) beleuchtet. Durch die Unterteilung können Schattengrenzen zwischen Dreieckseckpunkten genauer bestimmt werden.

Der Algorithmus zur Berechnung von Schattengrenzen mittels Distance Maps kann

auch in Verbindung mit Environment Maps verwendet werden. Dazu wäre es lediglich erforderlich, jedes Pixel der Environment Map als Lichtquelle für den Algorithmus aus 2.4 zu verwenden. Aus Performance-Gründen kann es aber sinnvoll sein, dazu eine Evironment Map von niedriger Auflösung zu verwenden oder sich auf die am hellsten strahlenden Pixel zu beschränken.

3.2 Berechnung des Lichtaustausches unter Berücksichtigung von BRDFs

Da Radiosity [Coh85, Gor84, Nis85] nur diffuse Interreflexionen modellieren kann, werden inzwischen für photorealistische Grafik meist stochastische Raytracing-Verfahren verwendet. Dies ist natürlich mit dem Nachteil verbunden, dass die Bildsynthese nicht in Echtzeit durchgeführt werden kann. Bei Radiosity hingegen konnte eine ermittelte Lösung der Rendering Equation aus wechselnder Perspektive in Echtzeit angezeigt werden.

Während eine Visibility Map Richtungen für einfallendes Licht codiert, verteilt die BRDF das Licht auf ausfallende Richtungen. Man könnte in einer zusätzlichen Map für jeden Knoten des Polygonnetzes speichern, wie sich reflektiertes Licht auf die verschiedenen Richtungen verteilt. Wir wollen diesen Map-Typus Reflectance Map nennen. Dabei können dieselben Abbildungen verwendet werden, die auch für Visibility Maps verwendet werden. Zur Berechnung der globalen Beleuchtung würde dann ein anderer Punkt auf die Reflection Map dieses Punktes zugreifen, um zu ermitteln, wieviel Licht in seine Richtung reflektiert wird.

Zur Lösung der Rendering Gleichung würde die Szene aufgeteilt in zwei Bereiche: im nahen Bereich würde die Szene durch Geometrie und BRDFs beschrieben. Nach außen hin könnte eine offene Szene durch eine High Dynamic Range Environment Map geschlossen werden, deren Pixel als Richtungslichtquellen interpretiert werden. Bei einem Raum könnte z.B. das Fenster durch ein High Dynamic Range Bild gegeben sein. Zusätzlich zu den Reflection Maps werden Visibility Maps benutzt, wie in dieser Arbeit beschrieben. Die Rendering Gleichung wird iterativ gelöst, wobei jede Iteration je eine Interreflexion zwischen allen Flächen der Szene berechnet. Nach der Initialisierung sind alle nicht emittierenden Flächen schwarz und die Reflection Maps der emittierenden Lichtquellen werden ermittelt. In jeder Iteration wird nun für jeden Knoten seine Visibility Map ausgewertet. Je nachdem, ob sich ein Pixel im Vorder- oder Hintergrund befindet, wird entweder auf die Environment Map zugegriffen, oder wir ermitteln aus der Reflection Map eines anderen Knotens das reflektierte Licht. Durch eine BRDF wird das von der Visibility Map empfangene Licht auf die Reflection Map verteilt. Nach mehreren Iterationen müsste eine gute Approximation der Rendering

Gleichung ermittelt worden sein. Dieses Verfahren ordnet jedem Vordergrundpixel p einer Visibility Map ein Polygon des Polygonnetzes als Lichtquelle $L_i(p)$ im Sinne von Gleichung (43) zu.

Beim Anzeigen der Ergebnisse muss für jeden Knoten aufgrund der Betrachterposition das richtige Pixel der Reflection Map des Knotens ausgewertet werden. Dies sollte jedoch mit interaktiven Framerates möglich sein, da wie bei der Berechnung von Schattengrenzen lediglich die Betrachterposition in die Reflection Map eines Punktes transformiert werden muss.

3.3 Schlussfolgerungen

Die vorliegende Arbeit hat Methoden zur Beleuchtung von Kleidung aufgezeigt. Die vorgestellten Algorithmen beruhen auf der Arbeit von Stewart [Ste99], gehen aber darüber hinaus. Die Visibility Maps erlauben wesentlich genauere Aussagen über die Umgebung eines Punktes als die Visibility Cones von Stewart, da die Visibility Cones nur an ihren Ecken exakt berechnet sind. Dazwischen können die Visibility Cones beliebig falsch sein. Ferner setzt sich diese Arbeit detailliert mit verschiedenen Arten von Lichtquellen auseinander.

Gegenüber Standardmethoden der Echtzeit-Computergrafik wurde Wert darauf gelegt, durch Verwendung möglichst vieler Lichtquellen flächige Lichtquellen nachzuahmen, wie sie in der Wirklichkeit überwiegen. Die existierenden Algorithmen zur Berechnung von Schatten in Echtzeit sind für die Verwendung mit vielen Lichtquellen nicht geeignet, auch wenn sie gute Ergebnisse für eine Lichtquelle liefern. Der Schattenvolumen-Algorithmus muss für jede Lichtquelle Schattenvolumen berechnen, während der z-Buffer-Algorithmus für jede Lichtquelle ein Bild der Szene benötigt. Der Aufwand für zusätzliche Lichtquellen ist hier so hoch, dass die in der Arbeit geschilderten Algorithmen wohl schneller sein werden.

Zum Vergleich: Bei einer Environment Map werden hingegen mindestens 256 Lichtquellen ausgewertet, und es werden im Programm MeshShader immer noch mehrere Frames pro Sekunde erzielt. Selbst wenn die Darstellung des kompletten Polygonnetzes nur 2,5 ms benötigt (vergleiche Tabelle 2), würde die Berechnung und Auswertung von 256 Schattenvolumen oder die Generierung von 256 z-Buffer Shadow Maps jeweils über 0,64s dauern. Die in dieser Arbeit vorgestellten Algorithmen sind also schneller im Falle von einer großen Anzahl von Lichtquellen. Für die hellsten Lichtquellen der Environment Map können zusätzlich Schattengrenzen berechnet werden, sodass die Darstellungsqualität noch weiter verbessert wird.

Obwohl die Algorithmen für die Schattierung von Kleidung entworfen wurden, sind die Erkenntnisse auf allgemeine Polygonnetze übertragbar. Die entworfenen Algorithmen eignen sich insbesondere für Polygonnetze mit vielen Falten und Konkavitäten, da die Algorithmen sehr viel Wert auf die Abschattung von Lichtquellen der Umgebung in Falten legen.

High Dynamic Range Bilder erlauben die Nachahmung der Beleuchtung an einem beliebigen Ort und ermöglichen Environment Mapping in Echtzeit unter Berücksichtigung von spekularen und diffusen Reflexionen, allerdings wird dabei Selbstabschattung vernachlässigt. Debevecs Technik hingegen erlaubt Selbstabschattung, kann aber nicht in Echtzeit laufen, da er einen Raytracer benutzt. Die neue Visibility Map-basierte Methode erlaubt Environment Mapping mit BRDFs und Selbstabschattung in Echtzeit.

A Kurzanleitung zum Demonstrationsprogramm

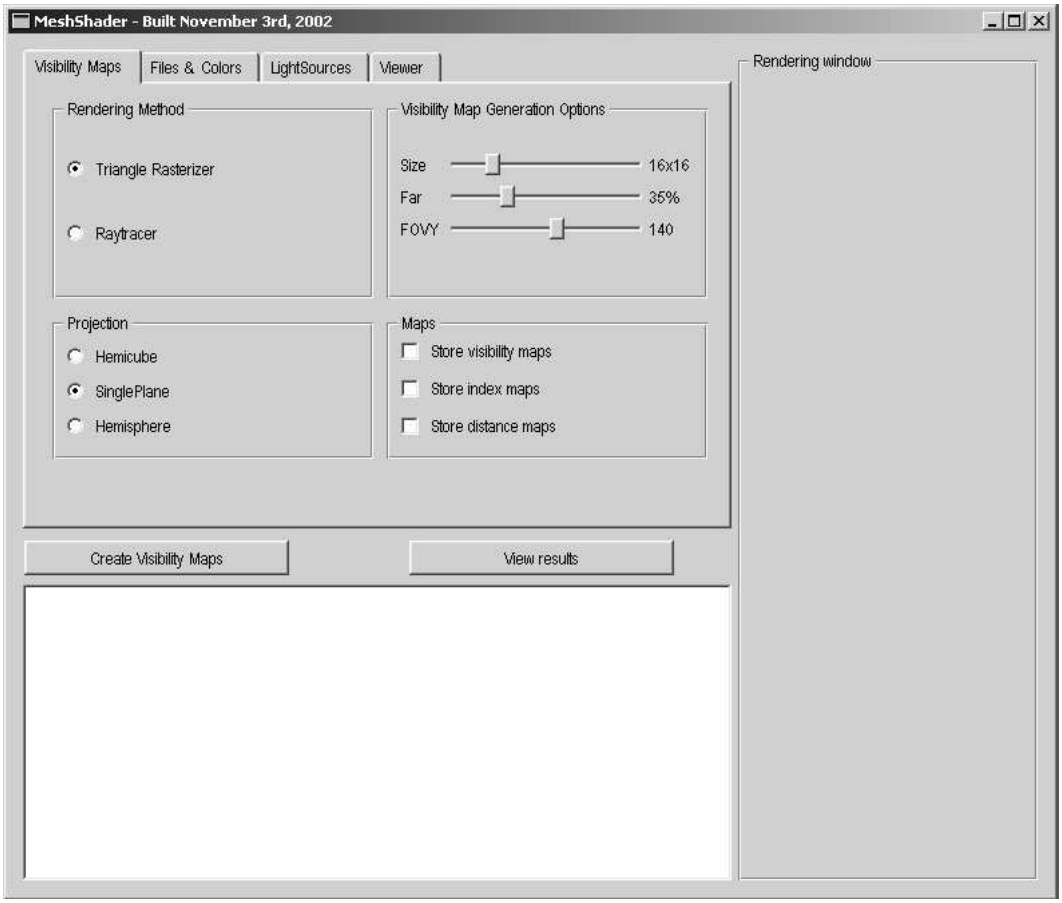

Abb.25: MeshShader Gesamtansicht

Nachdem das Programm MeshShader von [Gan02] heruntergeladen und in einem Verzeichnis entpackt wurde, kann es direkt in diesem Verzeichnis gestartet werden. Es erscheint ein dreiteiliger Dialog, siehe Abb. 25. Im oberen linken Teil werden verschiedene Einstellungen vorgenommen, der Bereich darunter zeigt Meldungen des Programms an. Im Bereich rechts daneben werden berechnete Visibility Maps angezeigt oder das Polygonnetz gerendert. Die Berechnung der Visibility Maps wird mit dem Button „Calculate Brightnesses" gestartet. Nachdem die Visibility Maps erzeugt wurden, können die Ergebnisse mit Hilfe des Buttons „View Results" angezeigt werden. Im folgenden werden die möglichen Einstellungen im linken oberen Fensterteil besprochen, und wie sie sich auf Berechnung und/oder Anzeige der Daten auswirken.

Im Tab „Rendering Method" kann einer der in Kapitel 2.1 besprochenen Algorithmen ausgewählt werden. Die Auswahl einer „Projection" wählt eine der in Kapitel 2.2 besprochenen Abbildungen aus. Die Einstellung „size" steuert die Größe der Visibility

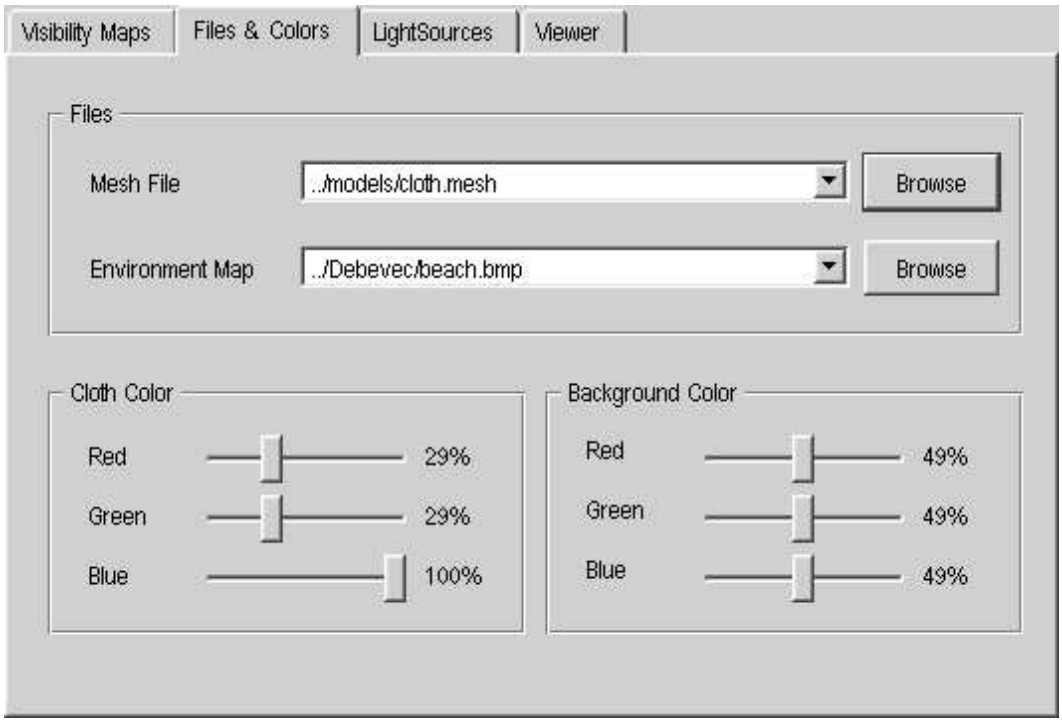

Abb.26: Tab 2 der Applikation

Map in Pixeln. „Far" steuert die Distanz der Far Clipping Plane, 100% entsprechen dem Durchmesser des Polygonnetzes. Bei kleinerer Distanz wird nur ein entsprechend kleinerer Teil des Polygonnetzes ausgewertet, wodurch die Generierung von Distance Maps beschleunigt wird. Die Einstellung FOVY bezieht sich ausschließlich auf die Abbildung SinglePlane, siehe Kapitel 2.2.2. Darunter kann gewählt werden, ob die erzeugten Visibility Maps gespeichert werden oder nach Auswertung verworfen werden. Zusätzlich können Distance Maps oder Index Maps gespeichert werden. Index Maps sind Visibility Maps, die zusätzliche Daten für den Zugriff auf Environment Maps enthalten. Für die Auswertung von Index Maps und Distance Maps werden gespeicherte Visibility Maps benötigt.

Nun zu den Optionen auf dem zweiten Tab, „Files & Colors" (Abb. 26). „Mesh File" enthält den Namen der Datei, für welche die Berechnungen durchgeführt werden sollen. Außerdem lässt sich die „Environment Map" wählen - es muss sich hierbei um eine Bitmap der Auflösung 1024×128 handeln. Bei animierten Environment Maps muss das erste Bitmap des Films angegeben werden, die restlichen Dateien werden vom Programm automatisch erkannt und nachgeladen. Durch die danebenliegenden Buttons können die Dateien über entsprechende Suchmasken gewählt werden. Durch Cloth Color und Ambient Color können die Farbe des Polygonnetzes und des Hintergrunds ausgewählt werden.

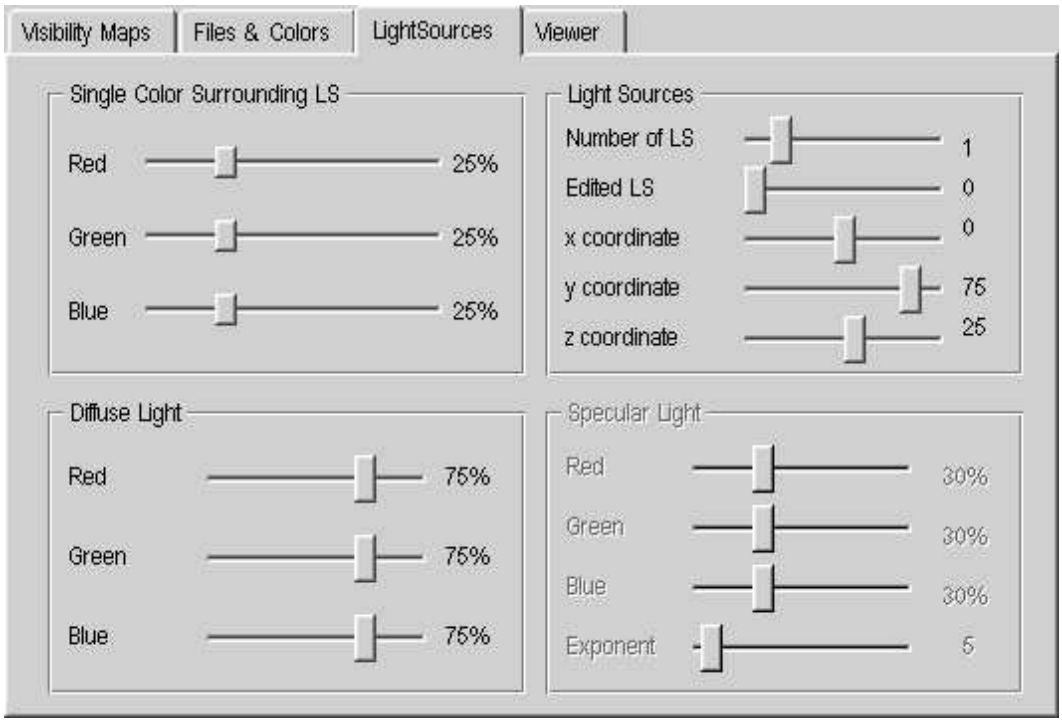

Abb.27: Tab 3 der Applikation

Der dritte Tab erlaubt Einstellungen für ambiente, diffuse und spekulare Lichtquellen (Abb. 27). Unter „Single Color Surrounding LS" lässt sich die Farbe einer umgebenden, einfarbigen Lichtquelle einstellen. Für diffuse und spekulare Lichtquellen läßt sich zusätzlich zur Farbe eine Position einstellen. Dazu wird mit „Number of LS" die Anzahl der Lichtquellen eingestellt und anschließend kann man mit „Edited LS" eine dieser Lichtquellen zum Editieren auswählen. Für die editierte Lichtquelle lassen sich nun Position und diffuse Bestrahlungsstärke einstellen.

Im letzten Tab kann man den Beleuchtungsmodus wählen. „Single Color Surrounding LS" bedeutet die Beleuchtung mit einer umgebenden, einfarbigen Lichtquelle. Der Modus „Point Light Sources and Shadows" beleuchtet das Polygonnetz mit Punktlichtquellen. Für diesen Modus lässt sich außerdem einstellen, in welcher Form Schattengrenzen angezeigt werden. Zur Verfügung stehen die Modi „Interpolated Shadows", die einfach zwischen Farbwerten interpoliert, „Shadows run halfway between Vertices", Schattengrenzen auf der Kantenmitte, sowie „Calculate Shadows from distance maps", berechnete Schattengrenzen. Für diese Modi müssen Visibility und Distance Maps gespeichert worden sein, siehe Beschreibung zu Tab 1. Die letzte Beleuchtungsoption, „Environment Map Lighting", erlaubt die Beleuchtung des Polygonnetzes mit (animierten) Environment Maps. Bei Verwendung von Punktlichtquellen oder Environment Maps kann mittels des Sliders „Lightsource Animation" eine Animation gestartet werden, oder mittels des Sliders „Rotation" eine Animationsphase gewählt wer-

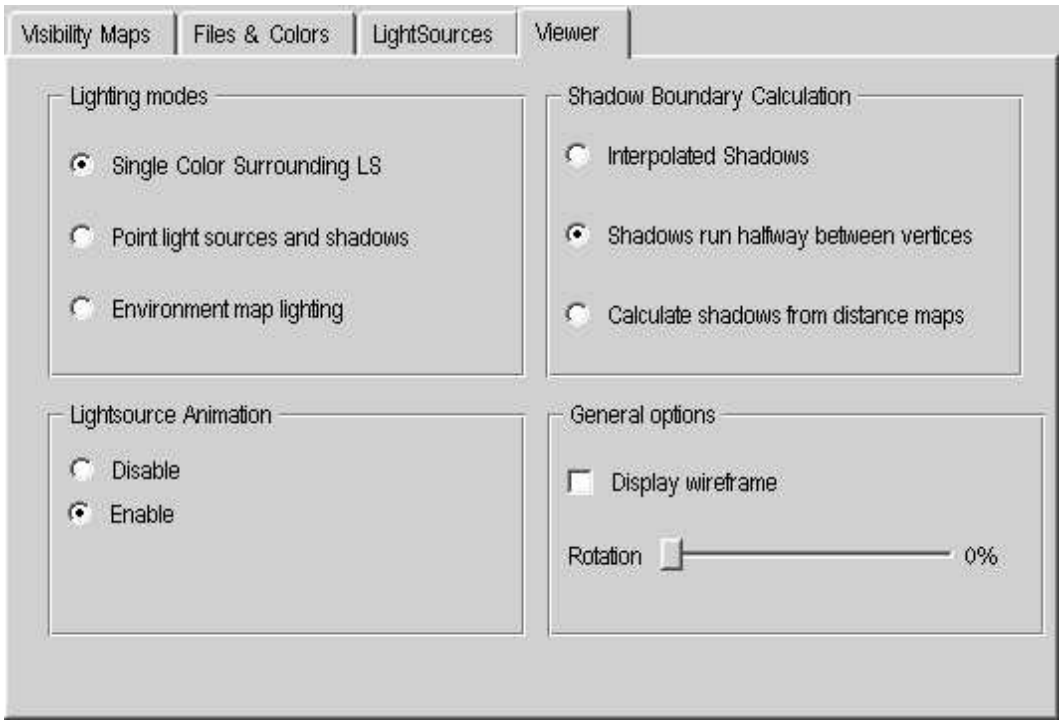

Abb.28: Tab 4 der Applikation

den. Die Option „Display Wireframe” erlaubt, zusätzlich den Wireframe des Polygonnetzes zu rendern.

Algorithmus	Abbildung	Auflösung	Zeit 1/s	Zeit 2/s	Zeit 3/s
Triangle	HC	8x8	4,7	5,0	?
Rasterizer	HC	16x16	4,8	5,7	?
(far = 100%)	HC	32x32	5,4	*	?
	HC	64x64	7,5	*	?
	SP	8x8	1,0	1,1	?
	SP	16x16	1,0	1,3	?
	SP	32x32	1,1	1,8	?
	SP	64x64	1,7	?	?
	SP	128x128	4,1	*	?
Raytracer	HS	16x4	4,0	4,2	5,9
	HS	32x8	15,2	15,6	23,2
	HS	64x16	59,3	61,1	91,4
	HS	128x32	234,7	236,1	374,7

Tabelle 4: Laufzeiten der verschiedenen Kombinationen von Algorithmen, Abbildung und Auflösung. *: hier reichte der Speicher nicht aus, **: hier trat eine Zeitabhängigkeit vom Öffnungswinkel auf, ?: Zeit wurde nicht bestimmt.

B Performance Messungen

Die in Tabelle 4 gemessenen *Laufzeiten* wurden auf einem Athlon-1800+ mit einer Nvidia GeForce4 Ti 4200 unter Linux in der Auflösung $1024 \times 64 \times 16$ anhand der Programmversion 2002-07-02 für die Datei `dress.obj` (vergleiche Abb. 10) ermittelt. Für die Distance Map-basierten Tests wurde Version 2002-11-03 verwendet, da 2002-07-02 Distance Maps nicht richtig bestimmt. Zeit 1 wird bei Aufnahme von lediglich der Visibility Map gemessen. Zeit 2 bezieht sich auf die Aufnahme einer Visibility Map mit farbcodierten Indizes in eine Environment Map. Zeit 3 tritt bei Aufnahme

Abbildung	Auflösung	Lichtquellen	Framerate
HC	8x8	192	31,0
HC	16x16	768	12,5
HC	32x32	3072	3,5
HS	16x4	64	41,0
HS	32x8	256	19,0
HS	64x16	1024	6,0

Tabelle 5: Framerate bei der Darstellung von dress.obj bei Beleuchtung mit Environment Maps

Algorithmus	Framerate in fps
Einfarbige, umgebende Lichtquelle	60,6
Interpolierte Schatten	35,2
Schatten in der Mitte	23,5
Schattengrenze aus Distance Map	14,3

Tabelle 6: Framerates bei der Darstellung von dress.obj mit einer Lichtquelle

von Visibility Maps mit anschließender Berechnung von Distance Maps auf.

Die Proportionalität der Raytracer Algorithmen zur Auflösung ist deutlich sichtbar. Bei den Triangle Rasterizer Algorithmen steigt die Laufzeit in Abhängigkeit von der Auflösung der Visibility Map in gerigerem Maße. Ebenfalls deutlich erkennbar ist der wesentlich größere Aufwand der Erzeugung der fünf HemiCube-Bilder im Vergleich zu den SinglePlane und HemiSphere Abbildungen, die nur ein Bild erfordern. In allen Algorithmen macht die Aufnahme der farbcodierten Environment Map nur einen geringen Unterschied für die Laufzeit. Dagegen verlängert die Bestimmung einer Distance Map die Erzeugung der Maps um etwa 50%. länger Als schnellster Algorithmus fällt Triangle Rasterizer mit SinglePlane auf. Hierbei muss jedoch der Zusammenhang zwischen größtem möglichen Öffnungswinkel und der Auflösung des Bildes berücksichtigt werden, damit die berechneten Farbwerte keine Artefakte aufweisen (2.2.2).

Tabelle 5 zeigt die Framerate bei der Beleuchtung mit Environment Maps, Tabelle 6 stellt die Framerates bei der Darstellung des Kleides dar.

Literatur

[App68] A. Appel, „*Some Techniques for Shading Machine Renderings of Solids*", SJCC, 1968, 37-45

[Beat82] J.C. Beatty and K.S. Booth, editors, „*Tutorial: Computer Graphics*", Second Edition, IEEE Computer Soc. Press, Silver Spring, 1982

[Ber86] Philippe Bergeron, „*A General Version of Crow's Shadow Volumes*", IEEE Computers and Applications, 1986

[Bli76] Jim Blinn, M.E. Newell, „*Texture and Reflection Mapping for Computer Generated Images*", CACM, 1976, ebenfalls in [Beat82]

[Bli88] Jim Blinn, „*Me and my (fake) shadow*", IEEE Computer Graphics and Applications, 1988, auch zu finden in „*Jim Blinn's Corner: A Trip Down the Graphics Pipeline*", 1996

[Bor02] Pavel Borodin, Marcin Novotni, Reinhard Klein, „*Progressive Gap Closing For Mesh Repairing*", Computer Graphics International 2002, abgedruckt in cite[Vince

[Bra02] Stefan Brabec, Thomas Annen, Hans-Peter Seidel, „*Shadow Mapping for Hemispherical and Omnidirectional Light Sources*", Computer Graphics International 2002, abgedruckt in John Vince, Rae Earnshaw (Hrsg.) „Advances in Modelling, Animation and Rendering", 2002

[Coh85] Cohen, Greenberg, „*The Hemi-Cube: A Radiosity Solution for Complex Environments*", Siggraph 1985

[Cro77] Franklin C. Crow, „*Shadow Algorithms for Computer Graphics*", Siggraph 1977, ebenfalls in [Beat82]

[Deb97] Paul Debevec, Jitendra Malik, „*Recovering High Dynamic Range Maps from Photographs*", Siggraph 1997

[Deb98] Paul Debevec, „*Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography*", Siggraph 1998

[Deb01] Paul Debevec, Dan Lemmon, „*Image-Based Lighting"*, Siggraph 2001 Course 14

[Enc97] José Encarnaçao, Wolfgang Straßer, Reinhard Klein, „*Graphische Datenverarbeitung*", in zwei Bänden, 4. Auflage, R. Oldenbourg Verlag München Wien 1997

[Eve02] Cass Everitt, Mark J. Kilgard, „*Practical & Robust Stencil Shadow Volumes for Hardware-Accelerated Rendering*", GDC 2002, zu finden unter www.developer.nvidia.com

[Gan01] Björn Ganster, Reinhard Klein, Mirko Sattler, Ralf Sarlette, „*Realtime Shading of Folded Surfaces*", Computer Graphics International 2002, abgedruckt in [Vin02]

[Gan02] Björn Ganster, *„MeshShader Homepage"*, zu finden unter cg.cs.uni-bonn.de/personal-pages/ganster

[Gla95] Andrew S. Glassner, „*Priciples of Digital Image Synthesis*", Band I, Morgan Kaufman Publishers Inc., 1995

[Gui87] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, R. E. Tarjan, „*Linear-time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons*", Algorithmica, 1987

[Gor84] Goral, Torrance, Greenberg, Battaile, „*Modeling the Interaction of Light Between Diffuse Surfaces*", Siggraph 1984

[Gou71] Gouraud, H., „*Continuous Shading of Curved Surfaces*", IEEE Transactions on Computers, 1971, ebenfalls abgedruckt in Freeman, „*Tutorial and Selected Readings in Interactive Computer Graphics*", IEEE Computer Society Press, 1980

[Hau02] Michael Hauth, Olaf Etzmuss, Bernd Eberhardt, Reinhard Klein, Ralf Sarlette, Mirko Sattler, Katja Daubert, Jan Kautz, „*Cloth Animation and Rendering*", Eurographics 2002

[Hei99] Wolfgang Heidrich, „*High-quality Shading and Lighting for Hardware-accelerateed Rendering*", Dissertation, 1999

[Hof99] Kenneth E. Hoff III, Tim Culver, John Keyser, Ming Lin, Dinesh Manocha, „*Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware*", Siggraph 1999, www.cs.unc.edu/ hoff/papers

[Kaj86] James T. Kajiya, „*The Rendering Equation*", Siggraph 1986

[Kil99] Mark J. Kilgard, „*Improving Shadows and Reflections via the Stencil Buffer*", 1999, zu finden unter www.developer.nvidia.com

[Kil01] Mark J. Kilgard, „*Shadow Mapping with Today's OpenGL Hardware*", CEDEC 2001, zu finden unter www.developer.nvidia.com

[Laf97] Eric P.F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, Donald P. Greenberg, „*Non-Linear Approximation of Reflectance Fields*", Siggraph 1997

[Meh99] Mehlhorn, Näher, „*LEDA - A platform for combinatorical and geometric computing*", Cambridge University Press, 1999

[Mil84] Gene S. Miller, „*Illumination and Reflectance Maps: Simulated Objects in Simulated and Real Scenes*", Siggraph 1984

[Nic77] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, T. Limpis, „Geometric considerations and nomenclature for reflectance", National Bureau of Standards, 1977

[Nis85] Nishita, Nakamae, „*Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection*", ACM Transactions On Graphics, 1985

[Ree87] William T. Reeves, David H. Salesin, Robert L. Cook, „*Rendering antialiased shadows with depth maps*", Siggraph 1987

[Pho75] Bui Tong, Phong, „*Illumination for Computer-Generated Pictures*", Communications of the ACM, 1975, ebenfalls in [Beat82]

[Seg02] Mark Segal, Kurt Akeley, „*The OpenGL Graphics System: A Specification (Version 1.4)*", 2002 zu finden unter www.opengl.org/developers/documentation/specs.html

[Shi00] Peter Shirley, „*Realistic Raytracing*", A K Peters Ltd, 2000

[Ste99] A. James Stewart, „*Computing Visibility from Folded Surfaces*", 1999, zu finden unter www.cs.queensu.ca/home/jstewart/chemise

[Sta02] Marc Stamminger, George Drettakis, „*Perspective Shadow Maps*", Siggraph 2002

[Tar88] R. J. Tarjan, C. J. van Wyck, „*An $O(n \log\log n)$-time algorithm for triangulating a simple polygon*", SIAM Journal on Computing, 1988

[Tor66] Kenneth E. Torrance, E.M. Sparrow, R.C. Birkebach, „*Polarization, directional distribution, and off-specular peak phenomena in light reflected from roughened surfaces*", Journal of the Optical Society of America, 1966

[Tor67] Kenneth E. Torrance, E.M. Sparrow, „*Theory for off-specular reflection from roughened surfaces*", Journal of the Optical Society of America, 1967

[Vin02] John Vince, Rae Earnshaw (Hrsg.), „*Advances in Modelling, Animation and Rendering*", Springer-Verlag, 2002

[War91] Gregory J. Ward, „*Real Pixels*", abgedruckt in James Harvo (Hrsg.), „*Graphics Gems II*, Academic Press, 1991

[War94] Gregory J. Ward, „*The Radiance Lighting Simulation and Rendering Program*", Siggraph 1994

[Whi80] Turner Whitted, „*An improved illumination model for shaded display*", Communications of the ACM, 1980

[Wil78] Lance Williams, „*Casting Curved Shadows on Curved Surfaces*", Siggraph 1978

Index

ambienten Term *5*
Appel Raytracing *32*
ausgedehnte Lichtquellen *2*
Bestrahlungsstärke *2*
Bildbasierte Beleuchtung *55*
Bogenmaß *1*
BRDF *3*
chag *40*
Cube Mapping *55*
Cube Visibility Map *38*
Diffuse Flächenlichtquelle *6*
Display List *20, 31*
Display List *31*
Environment Mapping *11, 55*
Environment Map *11*
Exponent *10*
Fluoreszenz *4*
Gitter *31*
Gleitkommazahl *10*
Goniometrische Lichtquelle *3*
Gouraud-Modell *6*
Halbschatten *13*
HemiCube *34*
HemiSphere *37*
High Dynamic Range *9, 55*
Hintergrundpixel *29*
Intensität *2*
Kernschatten *13*
Kleinwinkel-Näherung *48*
Konturkante *15*
Lambert-Strahler *6*
Laufzeiten *67*
Lichtquellen *2*
Low Dynamic Range *9*
Mach Band Effekt *7*
Phosphoreszenz *4*
Polarkoordinaten *37*
punktförmige Lichtquellen *2*
Punktlichtquelle *3, 13, 42*
Radiance *12*
Raumwinkel *1*
Raytracer *31*
Richtungslichtquelle *3, 55*
Schattenpolygon *15*
Schattenprojektion *13*
Schattenvolumen *15*
Schatten *13*
Selbstabschattung *48*
Shadow Map *19*
Sichtbarkeitskegel *22*
Signifikand *10*
Silhouettenpixel *29*
SinglePlane *35*
Spekulare Reflexion *6*
spezifische Ausstrahlung *1*
Strahldichte *2*
Strahler *3*
Strahlungsenergie *1*
Strahlungsleistung *1*
Triangle Rasterizer *30*
Triangle Strip List *31*
Triangle Strip *30*
Visibility Cones *22*
Visibility Maps *28*
Vordergrundpixel *29*
Voxel *31*
*z*Fail Schattenvolumen-Algorithmus *18*
*z*Pass Schattenvolumen-Algorithmus *17*

Danksagung

Bei Herr Prof. Klein möchte ich mich ganz herzlich bedanken bei für sein großes Interesse an dieser Diplomarbeit, seine vorzügliche Betreuung, die gemeinsamen Diskussionen und die Bereitstellung einer guten Arbeitsumgebung.

Bei Herr Prof. Weber bedanke ich mich für die Bereitschaft, diese Arbeit zu begutachten.

Meiner Freundin Astrid Schulte danke ich für ihre Unterstützung während der Diplomarbeit. Mirko Sattler, Ralf Sarlette, Doris Kiefer, Gerhard Bendels, Mark Lichtner und meinem Vater möchte ich für das Korrekturlesen der Arbeit danken. Meinen Eltern danke ich dafür, dass sie mir dieses Studium ermöglicht haben.

Mirko Sattler und Ralf Sarlette danke ich außerdem für produktive Diskussionen.

Ich bedanke mich bei Dr. Markus Wacker von der Universität Tübingen, der das Polygonnetz für das Kleid zur Verfügung gestellt hat.

Abschließend möchte ich mich bei allen Mitarbeitern der Computergrafik um Professor Klein für eine gute Zusammenarbeit und ein gutes Arbeitsklima bedanken.

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit

Effiziente Algorithmen
zur bildbasierten Beleuchtung
von Kleidung

selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Bonn, 7. November 2002

Björn Ganster